Manual de PHP

Stig Sæther Bakken

Alexander Aulbach

Egon Schmid

Jim Winstead

Lars Torben Wilson

Rasmus Lerdorf

Andrei Zmievski

Jouni Ahto

Editado por

Rafael Martínez (Coordinador)

Víctor Fernández

Leonardo Boshell

04-11-2002

Manual de PHP

por Stig Sæther Bakken, Alexander Aulbach, Egon Schmid, Jim Winstead, Lars Torben Wilson, Rasmus Lerdorf, Andrei Zmievski, y Jouni Ahto

Editado por Rafael Martínez (Coordinador), Víctor Fernández, y Leonardo Boshell

Publicado 04-11-2002

Tabla de contenidos

Prefacio

PHP, acrónimo de "PHP: Hypertext Preprocessor", es un lenguaje "Open Source" interpretado de alto nivel, especialmente pensado para desarrollos web y el cual puede ser embebido en páginas HTML. La mayoría de su sintaxis es similar a C, Java y Perl y es fácil de aprender. La meta de este lenguaje es permitir escribir a los creadores de páginas web, páginas dinámicas de una manera rápida y fácil, aunque se pueda hacer mucho más con PHP.

Este manual contiene principalmente una referencia de funciones PHP, tambien contiene una referencia del lenguaje, explicaciones de caracteristicas importantes de PHP y alguna información suplementaria.

Este manual se puede conseguir en diferentes formatos en http://www.php.net/docs.php. Estos ficheros son actualizados a medida que el manual vaya cambiando. Más información sobre como este manual es desarrollado puede encontrarse en el apéndice'Sobre este manual'

Parte I. Conceptos Básicos

Capítulo 1. Introducción

Qué es PHP?

PHP (acronimo de "PHP: Hypertext Preprocessor") es un lenguaje "open source" interpretado de alto nivel embebido en páginas HTML y ejecutado en el servidor.

Una respuesta corta y concisa, pero que significa realmente? Un ejemplo nos aclarará las cosas:

Ejemplo 1-1. Un ejemplo introductorio

```
<html>
    <head>
        <title>Example</title>
    </head>
    <body>

        <?php
        echo "Hi, I’m a PHP script!";
        ?>

    </body>
</html>
```

Podemos ver que no es lo mismo que un script escrito en otro lenguaje de programación como Perl o C -- En vez de escribir un programa con muchos comandos para crear una salida en HTML, escribimos el código HTML con cierto código PHP embebido (introducido) en el mismo, que producirá cierta salida (en nuestro ejemplo, producir un texto). El código PHP se incluye entre etiquetas especiales de comienzo y final que nos permitirán entrar y salir del modo PHP.

Lo que distingue a PHP de la tecnología Javascript, la cual se ejecuta en la máquina cliente, es que el código PHP es ejecutado en el servidor. Si tuviesemos un script similar al de nuestro ejemplo en nuestro servidor, el cliente sólamente recibiría el resultado de su ejecución en el servidor, sin ninguna posibilidad de determinar que código ha producido el resultado recibido. El servidor web puede ser incluso configurado para que procese todos los ficheros HTML con PHP.

Lo mejor de usar PHP es que es extremadamente simple para el principiante, pero a su vez, ofrece muchas caracteristicas avanzadas para los programadores profesionales. No tengais miedo de leer la larga lista de caracteristicas de PHP, en poco tiempo podreis empezar a escribir vuestros primeros scripts.

Aunque el desarrollo de PHP está concentrado en la programación de scripts en la parte del servidor, se puede utilizar para muchas otras cosas. Sigue leyendo y descubre más sobre PHP en la sección Qué se puede hacer con PHP?.

Qué se puede hacer con PHP?

PHP puede hacer cualquier cosa que se pueda hacer con un script CGI, como procesar la información de formularios, generar páginas con contenidos dinámicos, o mandar y recibir cookies. Y esto no es todo, se puede hacer mucho más.

Existen tres campos en los que scripts escritos en PHP son usados.

- Scripts en la parte del servidor. Este es el campo más tradicional y el principal campo de trabajo. Se necesitan tres cosas para que esto funcione. El parseador PHP (CGI ó módulo), un servidor web y un navegador. Se necesita correr el servidor web con PHP instalado. El resultado del programa PHP se puede obtener a través del navegador, conectando con el servidor web. Consultar la sección Instrucciones de instalación para más información.
- Scripts en linea de comandos. Podeis crear un script PHP y correrlo sin ningún servidor web ó navegador. Solamente necesitais el parseador PHP para usarlo de esta manera. Este tipo de uso es ideal para scripts ejecutados regularmente desde cron (en *nix ó Linux) ó el Planificador de tareas (en Windows). Estos scripts tambien pueden ser usados para tareas simples de procesado de texto. Consultar la sección Usos de PHP en la linea de comandos para más información.
- Escribir aplicaciones gráficas clientes. PHP no es probablemente el mejor lenguaje para escribir aplicaciones gráficas, pero si sabeis bien PHP, y os gustaria utilizar algunas características avanzadas en programas clientes, podeis utilizar PHP-GTK para escribir dichos programas. Es tambien posible escribir aplicaciones independientes de una plataforma. PHP-GTK es una extensión de PHP, no disponible en la distribución principal. Si te interesa PHP-GTK, puedes visitar las páginas web del projecto (http://gtk.php.net/).

PHP puede ser utilizado en cualquiera de los principales sistemas operativos del mercado, incluyendo Linux, muchas variantes Unix (incluido HP-UX, Solaris y OpenBSD), Microsoft Windows, Mac OS X, RISC OS y probablemente alguno más. PHP soporta la mayoría de servidores web de hoy en día, incluyendo Apache, Microsoft Internet Information Server, Personal Web Server, Netscape y iPlanet, Oreilly Website Pro server, Caudium, Xitami, OmniHTTPd y muchos otros. PHP tiene módulos disponibles para la mayoría de los servidores, para aquellos otros que soporten el estándar CGI, PHP puede usarse como procesador CGI.

Asi que, con PHP teneis la libertad de escoger el sistema operativo y el servidor de vuestro gusto. Tambien teneis la posibilidad de usar programación de procediminetos ó programación orientada a objetos. Aunque no todas la características estándares de la programación orientada a objetos están implementadas en la versión actual de PHP, muchas librerías y aplicaciones grandes (incluyendo la libreria PEAR) están escritas íntegramente usando programación orientada a objetos.

Con PHP no estais limitados a resultados en HTML. Entre las habilidades de PHP se incluyen, creación de imágenes, ficheros PDF y películas Flash (usando libswf y Ming) sobre la marcha. Tambien podeis presentar otros resultados, como XHTM y ficheros XML. PHP puede autogenerar estos ficheros y grabarlos en el sistema de ficheros en vez de presentarlos en la pantalla.

Quizas la característica más potente y destacable de PHP es su soporte para una gran cantidad de bases de datos. Escribir un interfaz via web para una base de datos es una tarea simple con PHP. Las siguientes bases de datos están soportadas actualmente:

Adabas D	Ingres	Oracle (OCI7 and OCI8)
dBase	InterBase	Ovrimos
Empress	FrontBase	PostgreSQL
FilePro (read-only)	mSQL	Solid
Hyperwave	Direct MS-SQL	Sybase
IBM DB2	MySQL	Velocis
Informix	ODBC	Unix dbm

Tambien tenemos una extensión DBX de abstracción de base de datos que permite usar de forma

transparente cualquier base de datos soportada por la extension. Adicionalmente, PHP soporta ODBC (The Open Database Connection standard), asi que podeis conectar a cualquier base de datos que soporte este estandar.

PHP tambien tiene soporte para comunicarse con otros servicios usando protocolos tales como LDAP, IMAP, SNMP, NNTP, POP3, HTTP, COM (en Windows) y muchos otros. Tambien se pueden crear raw sockets. PHP soporta WDDX para intercambio de datos entre lenguajes de programación en web. Y hablando de interconexión, PHP puede utilizar objetos Java de forma transparente como objetos PHP Y la extensión de CORBA puede ser utilizada para acceder a objetos remotos.

PHP tiene unas características muy útiles para el proceso de texto, desde expresiones regulares POSIX Extended ó Perl hasta parseador de documentos XML. Para parsear y acceder documentos XML, soportamos los estandares SAX y DOM. Podeis utilizar la extensión XSLT para transformar documentos XML.

Si usais PHP en el campo del comercio electrónico, encontrareis muy útiles las funciones Cybercash, CyberMUT, VeriSign Payflow Pro y CCVS para vuestros programas de pago.

Para terminar, tenemos muchas otras extensiones muy interesantes, las funciones del motor de búsquedas mnoGoSearch, funciones para pasarelas de IRC, utilidades de compresión (gzip, bz2),, conversión de calendarios, traducción .....

Como podeis ver esta página no es suficiente para enumerar todas las características y beneficios que PHP ofrece. Consultar las secciones Instalando PHP y Referencia de las funciones para una explicación de las extensiones mencionadas aqui.

Capítulo 2. A simple tutorial

Here we would like to show the very basics of PHP in a short simple tutorial. This text only deals with dynamic webpage creation with PHP, though PHP is not only capable of creating webpages. See the section titled What can PHP do for more information.

PHP-enabled web pages are treated just like regular HTML pages and you can create and edit them the same way you normally create regular HTML pages.

What do I need?

In this tutorial we assume that your server has support for PHP activated and that all files ending in `.php` are handled by PHP. On most servers this is the default extension for PHP files, but ask your server administrator to be sure. If your server supports PHP then you don't need to do anything. Just create your `.php` files and put them in your web directory and the server will magically parse them for you. There is no need to compile anything nor do you need to install any extra tools. Think of these PHP-enabled files as simple HTML files with a whole new family of magical tags that let you do all sorts of things.

Your first PHP-enabled page

Create a file named `hello.php` under your webserver root directory with the following content:

Ejemplo 2-1. Our first PHP script: `hello.php`

```
<html>
 <head>
  <title>PHP Test</title>
 </head>
 <body>
 <?php echo "Hello World<p>"; ?>
 </body>
</html>
```

The output of this script will be:

```
<html>
 <head>
  <title>PHP Test</title>
 </head>
 <body>
 Hello World<p>
 </body>
</html>
```

Note that this is not like a CGI script. The file does not need to be executable or special in any way. Think of it as a normal HTML file which happens to have a set of special tags available to you that do a lot of interesting things.

This program is extremely simple and you really didn't need to use PHP to create a page like this. All it does is display: `Hello World` using the PHP echo() statement.

If you tried this example and it didn't output anything, or it prompted for download, or you see the whole file as text, chances are that the server you are on does not have PHP enabled. Ask your administrator to enable it for you using the Installation chapter of the manual. If you want to develop PHP scripts locally, see the downloads (http://www.php.net/downloads.php) section. You can develop locally on any Operating system, be sure to install an appropriate web server too.

The point of the example is to show the special PHP tag format. In this example we used `<?php` to indicate the start of a PHP tag. Then we put the PHP statement and left PHP mode by adding the closing tag, `?>`. You may jump in and out of PHP mode in an HTML file like this all you want.

> **A Note on Text Editors:** There are many text editors and Integrated Development Environments (IDEs) that you can use to create, edit and manage PHP files. A partial list of these tools is maintained at PHP Editor's List (http://www.itworks.demon.co.uk/phpeditors.htm). If you wish to recommend an editor, please visit the above page and ask the page maintainer to add the editor to the list.

> **A Note on Word Processors:** Word processors such as StarOffice Writer, Microsoft Word and Abiword are not good choices for editing PHP files.
>
> If you wish to use one for this test script, you must ensure that you save the file as PLAIN TEXT or PHP will not be able to read and execute the script.

> **A Note on Windows Notepad:** If you are writing your PHP scripts using Windows Notepad, you will need to ensure that your files are saved with the .php extension. (Notepad adds a .txt extension to files automatically unless you take one of the following steps to prevent it.)
>
> When you save the file and are prompted to provide a name for the file, place the filename in quotes (i.e. "hello.php").
>
> Alternately, you can click on the 'Text Documents' drop-down menu in the save dialog box and change the setting to "All Files". You can then enter your filename without quotes.

Something Useful

Let's do something a bit more useful now. We are going to check what sort of browser the person viewing the page is using. In order to do that we check the user agent string that the browser sends as part of its HTTP request. This information is stored in a variable. Variables always start with a dollar-sign in PHP. The variable we are interested in right now is `$_SERVER["HTTP_USER_AGENT"]`.

> **PHP Autoglobals Note:** $_SERVER is a special reserved PHP variable that contains all web server information. It's known as an Autoglobal (or Superglobal). See the related manual page on Autoglobals for more information. These special variables were introduced in PHP 4.1.0 (http://www.php.net/release_4_1_0.php). Before this time, we used the older `$HTTP_*_VARS` arrays instead, such as `$HTTP_SERVER_VARS`. Although deprecated, these older variables still exist. (See also the note on old code.)

To display this variable, we can simply do:

Ejemplo 2-2. Printing a variable (Array element)

```
<?php echo $_SERVER["HTTP_USER_AGENT"]; ?>
```

A sample output of this script may be:

```
Mozilla/4.0 (compatible; MSIE 5.01; Windows NT 5.0)
```

There are many types of variables available in PHP. In the above example we printed an Array element. Arrays can be very useful.

`$_SERVER` is just one variable that's automatically made available to you by PHP. A list can be seen in the Reserved Variables section of the manual or you can get a complete list of them by creating a file that looks like this:

Ejemplo 2-3. Show all predefined variables with phpinfo()

```
<?php phpinfo(); ?>
```

If you load up this file in your browser you will see a page full of information about PHP along with a list of all the variables available to you.

You can put multiple PHP statements inside a PHP tag and create little blocks of code that do more than just a single echo. For example, if we wanted to check for Internet Explorer we could do something like this:

Ejemplo 2-4. Example using control structures and functions

```
<?php
if (strstr($_SERVER["HTTP_USER_AGENT"], "MSIE")) {
```

```
echo "You are using Internet Explorer<br />";
}
?>
```

A sample output of this script may be:

```
You are using Internet Explorer<br />
```

Here we introduce a couple of new concepts. We have an if statement. If you are familiar with the basic syntax used by the C language this should look logical to you. If you don't know enough C or some other language where the syntax used above is used, you should probably pick up any introductory PHP book and read the first couple of chapters, or read the Language Reference part of the manual. You can find a list of PHP books at http://www.php.net/books.php.

The second concept we introduced was the strstr() function call. strstr() is a function built into PHP which searches a string for another string. In this case we are looking for `"MSIE"` inside `$_SERVER["HTTP_USER_AGENT"]`. If the string is found, the function returns `TRUE` and if it isn't, it returns `FALSE`. If it returns `TRUE`, the if statement evaluates to `TRUE` and the code within its {braces} is executed. Otherwise, it's not. Feel free to create similar examples, with if, else, and other functions such as strtoupper() and strlen(). Each related manual page contains examples too.

We can take this a step further and show how you can jump in and out of PHP mode even in the middle of a PHP block:

Ejemplo 2-5. Mixing both HTML and PHP modes

```
<?php
if (strstr($_SERVER["HTTP_USER_AGENT"], "MSIE")) {
?>
<h3>strstr must have returned true</h3>
<center><b>You are using Internet Explorer</b></center>
<?php
} else {
?>
<h3>strstr must have returned false</h3>
<center><b>You are not using Internet Explorer</b></center>
<?php
}
?>
```

A sample output of this script may be:

```
<h3>strstr must have returned true</h3>
```

```
<center><b>You are using Internet Explorer</b></center>
```

Instead of using a PHP echo statement to output something, we jumped out of PHP mode and just sent straight HTML. The important and powerful point to note here is that the logical flow of the script remains intact. Only one of the HTML blocks will end up getting sent to the viewer depending on if strstr() returned `TRUE` or `FALSE` In other words, if the string `MSIE` was found or not.

Dealing with Forms

One of the most powerful features of PHP is the way it handles HTML forms. The basic concept that is important to understand is that any form element in a form will automatically be available to your PHP scripts. Please read the manual section on Variables from outside of PHP for more information and examples on using forms with PHP. Here's an example HTML form:

Ejemplo 2-6. A simple HTML form

```
<form action="action.php" method="POST">
 Your name: <input type="text" name="name" />
 Your age: <input type="text" name="age" />
 <input type="submit">
</form>
```

There is nothing special about this form. It is a straight HTML form with no special tags of any kind. When the user fills in this form and hits the submit button, the `action.php` page is called. In this file you would have something like this:

Ejemplo 2-7. Printing data from our form

```
Hi <?php echo $_POST["name"]; ?>.
You are <?php echo $_POST["age"]; ?> years old.
```

A sample output of this script may be:

```
Hi Joe.
You are 22 years old.
```

It should be obvious what this does. There is nothing more to it. The `$_POST["name"]` and `$_POST["age"]` variables are automatically set for you by PHP. Earlier we used the `$_SERVER` autoglobal, now above we just introduced the $_POST autoglobal which contains all POST data. Notice how the *method* of our form is POST. If we used the method *GET* then our form information would live in the $_GET autoglobal instead. You may also use the $_REQUEST autoglobal if you don't care the source of your request data. It contains a mix of GET, POST, COOKIE and FILE data. See also the import_request_variables() function.

Using old code with new versions of PHP

Now that PHP has grown to be a popular scripting language, there are more resources out there that have listings of code you can reuse in your own scripts. For the most part the developers of the PHP language have tried to be backwards compatible, so a script written for an older version should run (ideally) without changes in a newer version of PHP, in practice some changes will usually be needed.

Two of the most important recent changes that affect old code are:

- The deprecation of the old `$HTTP_*_VARS` arrays (which need to be indicated as global when used inside a function or method). The following autoglobal arrays were introduced in PHP 4.1.0 (http://www.php.net/release_4_1_0.php). They are: `$_GET`, `$_POST`, `$_COOKIE`, `$_SERVER`, `$_ENV`, `$_REQUEST`, and `$_SESSION`. The older `$HTTP_*_VARS` arrays, such as $HTTP_POST_VARS, still exist and have since PHP 3.
- External variables are no longer registered in the global scope by default. In other words, as of PHP 4.2.0 (http://www.php.net/release_4_2_0.php) the PHP directive register_globals is *off* by default in `php.ini`. The preferred method of accessing these values is via the autoglobal arrays mentioned above. Older scripts, books, and tutorials may rely on this directive being on. If on, for example, one could use `$id` from the URL `http://www.example.com/foo.php?id=42`. Whether on or off, `$_GET['id']` is available.

For more details on these changes, see the section on predefined variables and links therein.

What's next?

With what you know now you should be able to understand most of the manual and also the various example scripts available in the example archives. You can also find other examples on the php.net websites in the links section: http://www.php.net/links.php.

Capítulo 3. Instalación

Bajándose la última versión

El código fuente y las distribuciones binarias para algunas plataformas (incluído Windows) se pueden encontrar en `http://www.php.net/`.

Instalación en sistemas UNIX

Esta sección le guiará a través de la configuración e instalación del PHP. Conocimientos y software necesarios:

- Habilidades básicas en UNIX (ser capaz de manejar el "make" y un compilador de C)
- Un compilador ANSI de C
- Un servidor web

Instrucciones Rápidas de Instalación (Versión Módulo de Apache)

```
1.  gunzip apache_1.3.x.tar.gz
2.  tar xvf apache_1.3.x.tar
3.  gunzip php-3.0.x.tar.gz
4.  tar xvf php-3.0.x.tar
5.  cd apache_1.3.x
6.  ./configure --prefix=/www
7.  cd ../php-3.0.x
8.  ./configure --with-mysql --with-apache=../apache_1.3.x --enable-track-vars
9.  make
10. make install
11. cd ../apache_1.3.x
12. ./configure --prefix=/www --activate-module=src/modules/php3/libphp3.a
13. make
14. make install

  En lugar de este paso quizás prefiera simplemente copiar el binario
  httpd encima del binario existente. Si lo hace, asegúrese antes de
  cerrar su servidor.

15. cd ../php-3.0.x
16. cp php3.ini-dist /usr/local/lib/php3.ini

  Puede editar el archivo /usr/local/lib/php3.ini para ajustar opciones
  del PHP. Si prefiere tenerlo en otro sitio, utilice
  --with-config-file-path=/path en el paso 8.

17. Edite su archivo httpd.conf o srm.conf y añada:

          AddType application/x-httpd-php3 .php3
```

```
Puede elegir la extensión que desee aquí. .php3 es simplemente nuestra
sugerencia.
```

18. `Utilice su método habitual para iniciar el servidor Apache (debe detener y reiniciar el servidor, no solamente hacerlo recargarse usando una señal HUP o USR1.)`

Configuración

Hay dos maneras de configurar el PHP.

- Utilizando el script de "setup" que viene con el PHP. Este script le hace una serie de preguntas (casi como el script "install" del PHP/FI 2.0) y ejecuta el "configure" al final. Para ejecutar este script, escriba **./setup**.

  Este script también creará un archivo llamado "do-conf", que contendrá las opciones pasadas a la configuración. Puede editar este archivo para cambiar algunas opciones sin tener que re-ejecutar el "setup". Escriba luego **./do-conf** para ejecutar la configuración con las nuevas opciones.

- Ejecutar el "configure" a mano. Para ver las opciones de que dispone, escriba **./configure --help**.

Los detalles sobre las distintas opciones de configuración son listados a continuación.

Módulo del Apache

Para configurar el PHP como módulo de Apache, responda "yes" a "Build as an Apache module?" (la opción --with-apache=*DIR* es la que lo configura) y especifique el directorio base de la distribución de Apache. Si ha desempacado el Apache en `/usr/local/www/apache_1.2.4`, este será su directorio base de la distribución de Apache. El directorio por defecto es `/usr/local/etc/httpd`.

Módulo fhttpd

Para configurar el PHP como módulo fhttpd, responda "yes" a "Build as an fhttpd module?" (la opción --with-fhttpd=*DIR* es la que lo configura) y especifique el directorio base del fuente del fhttpd. El directorio por defecto es `/usr/local/src/fhttpd`. Si está ejecutando fhttpd, configurar PHP como módulo le dará mejor rendimiento, más control y capacidad de ejecución remota.

CGI version

El valor por defecto es configurar el PHP como programa CGI. Si está ejecutando un servidor web para el que el PHP tiene soporte como módulo, debería elegir dicha solución por motivos de rendimiento. Sin embargo, la versión CGI permite a los usuarios del Apache el ejecutar distintas páginas con PHP bajo

distintos identificadores de usuario. Por favor, asegúrese de haber leído el capítulo sobre Seguridad si va a ejecutar el PHP como CGI.

Opciones de soporte para Base de Datos

El PHP tiene soporte nativo para bastantes bases de datos (así como para ODBC):

Adabas D

```
--with-adabas=DIR
```

Compila con soporte para Adabas D. El parámetro es el directorio de instalación de Adabas D y por defecto vale `/usr/local/adabasd`.

Página de Adabas (http://www.adabas.com/)

dBase

```
--with-dbase
```

Habilita el soporte integrado para DBase. No se precisan librerías externas.

filePro

```
--with-filepro
```

Habilita el soporte integrado de sólo lectura para filePro. No se precisan librerías externas.

mSQL

```
--with-msql=DIR
```

Habilita el soporte para mSQL. El parámetro es el directorio de instalación de mSQL y por defecto vale `/usr/local/Hughes`. Este es el directorio por defecto de la distribución mSQL 2.0. **configure** detecta automáticamente qué versión de mSQL está ejecutándose y el PHP soporta tanto 1.0 como 2.0, pero si compila el PHP con mSQL 1.0 sólo podrá acceder a bases de datos de esa versión y viceversa.

Vea también Directivas de Configuración de mSQL en el archivo de configuración.

Página de mSQL (http://www.hughes.com.au)

MySQL

```
--with-mysql=DIR
```

Habilita el soporte para MySQL. El parámetro es el directorio de instalación de MySQL y por defecto vale `/usr/local`. Este es el directorio de instalación de la distribución de MySQL.

Vea también Directivas de Configuración de MySQL en el archivo de configuración.

Página de MySQL (http://www.tcx.se)

iODBC

```
--with-iodbc=DIR
```

Incluye soporte para iODBC. Esta característica se desarrolló inicialmente para el iODBC Driver Manager, un gestor de controlador de ODBC de redistribución libre que ese ejecuta bajo varios sabores de UNIX. El parámetro es el directorio de instalación de iODBC y por defecto vale `/usr/local`.

Página de FreeODBC (http://users.ids.net/~bjepson/freeODBC/) o página de iODBC (http://www.iodbc.org)

OpenLink ODBC

```
--with-openlink=DIR
```

Incluye soporte para OpenLink ODBC. El parámetro es el directorio de instalación de OpenLink ODBC y por defecto vale `/usr/local/openlink`.

Página de OpenLink Software (http://www.openlinksw.com/)

Oracle

```
--with-oracle=DIR
```

Incluye soporte para Oracle. Se ha probado y debería funcionar al menos con las versiones de la 7.0 a la 7.3. El parámetro es el directorio ORACLE_HOME. No necesita especificar este parámetro si su entorno de Oracle ya está ajustado.

Página de Oracle (http://www.oracle.com)

PostgreSQL

```
--with-pgsql=DIR
```

Incluye soporte para PostgreSQL. El parámetro es el directorio base de la instalación de PostgreSQL y por defecto vale `/usr/local/pgsql`.

Vea también Directivas de Configuración de Postgres en el archivo de configuración.

Página de PostgreSQL (http://www.postgreSQL.org/)

Solid

```
--with-solid=DIR
```

Incluye soporte para Solid. El parámetro es el directorio de instalación y vale por defecto `/usr/local/solid`.

Página de Solid (http://www.solidtech.com)

Sybase

```
--with-sybase=DIR
```

Incluye soporte para Sybase. El parámetro es el directorio de instalación y vale por defecto `/home/sybase`.

Vea también Directivas de Configuración de Sybase en el archivo de configuración.

Página de Sybase (http://www.sybase.com)

Sybase-CT

```
--with-sybase-ct=DIR
```

Incluye soporte para Sybase-CT. El parámetro es el directorio de instalación de Sybase-CT y por defecto vale `/home/sybase`.

Vea también Directovas de Configuración de Sybase-CT en el archivo de configuración.

Velocis

```
--with-velocis=DIR
```

Incluye soporte para Velocis. El parámetro es el directorio de instalación de Velocis y vale por defecto `/usr/local/velocis`.

Página de Velocis (http://www.raima.com)

Una librería a medida de ODBC

```
--with-custom-odbc=DIR
```

Incluye soporte para una librería a medida arbitraria de ODBC. El parámetro es el directorio base y por defecto vale `/usr/local`.

Esta opción implica que se ha definido CUSTOM_ODBC_LIBS cuando se ejecutó el script de configuración. También deberá tener una cabecera odbc.h válida en algún lugar de su sendero (path) de inclusión. Si no tiene uno, créelo e incluya su cabecera específica desde ahí. Su cabecera puede requerir algunas definiciones extra, particularmente si es multiplataforma. Defínalas en CFLAGS.

Por ejemplo, puede usar Sybase SQL Anywhere bajo QNX como sigue: `CFLAGS=-DODBC_QNX LDFLAGS=-lunix CUSTOM_ODBC_LIBS="-ldblib -lodbc" ./configure --with-custom-odbc=/usr/lib/sqlany50`

ODBC Unificado

```
--disable-unified-odbc
```

Deshabilita el módulo de ODBC Unificado, que es un interfaz común a todas las bases de datos con interfaces basados en ODBC, tales como Solid y Adabas D. También funciona para librerías normales de ODBC. Ha sido probado con iODBC, Solid, Adabas D y Sybase SQL Anywhere. Requiere que uno (y sólo uno) de estos módulos o el módulo de Velocis esté habilitado, o que se especifique una librería a medida de ODBC. Esta opción sólo se puede aplicar si alguna de estas opciones es usada: --with-iodbc, --with-solid, --with-adabas, --with-velocis, o --with-custom-odbc.

Vea también Directivas de Configuración de ODBC Unificado en el archivo de configuración.

LDAP

```
--with-ldap=DIR
```

Incluye soporte para LDAP (Lightweight Directory Access Protocol - Protocolo Ligero de Acceso a Directorios). El parámetro es el directorio base de instalación de LDAP, y por defecto vale `/usr/local/ldap`.

Puede encontrar más información sobre LDAP en RFC1777 (ftp://ftp.isi.edu/in-notes/rfc1777.txt) y en RFC1778 (ftp://ftp.isi.edu/in-notes/rfc1778.txt).

Otras opciones de configuración

--with-mcrypt=*DIR*

```
--with-mcrypt
```

Incluye soporte para la librería mcrypt. Vea la documentación de mcrypt para más información. Si utiliza el argumento opcional *`DIR`*, el PHP buscará mcrypt.h en *`DIR`*/include.

--enable-sysvsem

```
--enable-sysvsem
```

Incluye soporte para semáforos Sys V (soportados por muchos derivados Unix). Vea la documentación sobre Semáforos y Memoria Compartida para más información.

--enable-sysvshm

```
--enable-sysvshm
```

Incluye soporte para la memoria compartida Sys V (soportada por muchos derivados Unix). Vea la documentación sobre Semáforos y Memoria Compartida para más información.

--with-xml

```
--with-xml
```

Incluye soporte para un parser XML no validador que utiliza la librería expat (http://www.jclark.com/xml/) de James Clark. Vea la referencia de funciones XML para más detalles.

--enable-maintainer-mode

```
--enable-maintainer-mode
```

Activa avisos extra de dependencias y del compilador utilizados por algunos de los desarrolladores del PHP.

--with-system-regex

```
--with-system-regex
```

Utiliza la librería de expresiones regulares del sistema en lugar de la incluída. Si está compilando PHP como módulo de servidor, debe utilizar la misma librería cuando genere el PHP y cuando lo enlace con el servidor. Active esto si la librería del sistema proporciona características especiales que pueda necesitar. Se recomienda utilizar la librería incluída siempre que sea posible.

--with-config-file-path

```
--with-config-file-path=DIR
```

El path utilizado para buscar el archivo de configuración cuando arranca el PHP.

--with-exec-dir

```
--with-exec-dir=DIR
```

Sólo permite ejecutar programas en DIR cuando está en modo seguro. Por defecto vale `/usr/local/bin`. Esta opción sólo fija el valor por defecto. Puede ser cambiado posteriormente mediante la directiva safe_mode_exec_dir en el fichero de configuración .

--enable-debug

```
--enable-debug
```

Habilita información de depuración adicional. Esto hace posible obtener información más detallada cuando hay problemas con el PHP. (Nótese que esto no tiene que ver con las facilidades de depuración o con la información disponible para los script PHP).

--enable-safe-mode

```
--enable-safe-mode
```

Habilita el "modo seguro" por defecto. Esto impone varias restricciones sobre lo que el PHP puede hacer, tales como abrir fichero sólo en el raiz de documentos. Lea el capítulo de Seguridad para más información. Los usuarios de CGI deberán siempre habilitar el modo seguro. Esta opción sólo fija el valor por defecto. Puede ser habilitado o deshabilitado posteriormente mediante la directiva safe_mode en el archivo de configuración.

--enable-track-vars

```
--enable-track-vars
```

Hace que el PHP lleve el control de dónde proceden las variables GET/POST/cookie usando las matrices HTTP_GET_VARS, HTTP_POST_VARS y HTTP_COOKIE_VARS. Esta opción sólo fija el valor por defecto. Puede ser habilitado o deshabilitado posteriormente mediante la directiva track_vars en el archivo de configuración.

--enable-magic-quotes

```
--enable-magic-quotes
```

Habilita las comillas mágicas por defecto. Esta opción sólo fija el valor por defecto. Puede ser habilitada o deshabilitada posteriormente mediante la directiva magic_quotes_runtime en el archivo de configuración. Vea también las directivas magic_quotes_gpc y magic_quotes_sybase.

--enable-debugger

```
--enable-debugger
```

Habilita el soporte de depuración interno del PHP. Esta característica aún está en estado experimental. Vea también las directivas de Configuración del Depurador en el archivo de configuración.

--enable-discard-path

```
--enable-discard-path
```

Si está habilitado, el ejecutable CGI del PHP se puede situar tranquilamente fuera del árbol de la web y la gente no podrá saltarse la seguridad del .htaccess. Lea la sección en el capítulo de seguridad sobre esta opción.

--enable-bcmath

```
--enable-bcmath
```

Habilita las funciones matemáticas de precisión arbitraria estilo **bc**. Vea también la opción bcmath.scale en el archivo de configuración.

--enable-force-cgi-redirect

```
--enable-force-cgi-redirect
```

Habilita la comprobación de seguridad para redirecciones internas del servidor. Deberá usar esta opción si está ejecutando la versión CGI bajo Apache.

Cuando se utiliza el PHP como un ejecutable CGI, siempre comprueba primero is está siendo utilizado bajo redirección (por ejemplo bajo Apache, usando directivas Action). Esto asegura que el ejecutable del PHP no se puede usar para saltarse los mecanismos estánder de autentificación del servidor web llamando al ejecutale directamente, como en `http://my.host/cgi-bin/php/secret/doc.html`. Este ejemplo accede al archivo `http://my.host/secret/doc.html` pero sin respetar ningún ajuste de seguridad del httpd para el directorio `/secret`.

No habilitando esta opción se deshabilita la comprobación y se permite el saltarse los ajustes de seguridad y autenticación del httpd. Haga esto sólo si el software de su servidor no puede indicar que se ha realizado una redirección segura y que todos sus archivos bajo la raiz de documentos y los directorios de los usuarios pueden ser accedidos por cualquiera.

Lea la sección en el capítulo de seguridad acerca de esta opción.

--disable-short-tags

```
--disable-short-tags
```

Deshabilita las etiquetas de PHP en formato corto `<? ?>`. Debe deshabilitar el formato corto si desea usar PHP con XML. Con el formato corto deshabilitado, la única etiqueta de código de PHP es `<?php ?>`. Esta opción sólo fija el valor por defecto. Puede ser habilitada o deshabilitada posteriormente mediante la directiva short_open_tag en el archivo de configuración.

--enable-url-includes

```
--enable-url-includes
```

Hace posible ejecutar código en otros servidores HTTP o FTP directamente desde el PHP usando include(). Vea también la opción include_path en el archivo de configuración.

--disable-syntax-hl

```
--disable-syntax-hl
```

Desconecta el resalte de sintáxis.

CPPFLAGS y LDFLAGS

Para hacer que la instalación de PHP busque los archivos de cabecera o de librería en distintos directorios, modifique las variables de entorno CPPFLAGS y LDFLAGS respectivamente. Si está utilizando un shell "sensible", podrá ejecutar **LDFLAGS=-L/my/lib/dir CPPFLAGS=-I/my/include/dir ./configure**

Construyendo

Cuando el PHP está configurado, ya está listo para construir el ejecutable CGI o la librería PERL. El comando **make** debería ocuparse de esto. Si fallara y no puede saber el motivo, vea la sección de Problemas.

Probando

Si ha construído el PHP como un programa CGI, puede probar su funcionamiento tecleando **make test**. Siempre es buena idea probar su construcción. Así puede atrapar pronto los problemas del PHP en su plataforma sin tener que batallar con ellos luego.

Comprobando la velocidad

Si ha construído el PHP como un programa CGI, puede comprobar la velocidad de su código escribiendo **make bench**. Nótese que se el modo seguro está habilitado por defecto, el test no podrá finalizar si se toma más de los 30 segundos disponibles. Esto se debe a que la función set_time_limit() no se puede usar en modo seguro. Use el ajuste de configuración max_execution_time para controlar este tiempo en sus propios script. **make bench** ignora el archivo de configuración.

Instalación en sistemas Windows 95/98/NT

Esta guía de instalación le ayudará a instalar y configurar el PHP en sus servidores web bajo Windows 9x/NT. Esta guía fue compilada por Bob Silva (mailto:bob_silva@mail.umesd.k12.or.us). La última revisión puede encontrarse en http://www.umesd.k12.or.us/php/win32install.html.

Esta guía proporciona soporte de instalacion para:

- Personal Web Server (se recomienda la última versión)
- Internet Information Server 3 ó 4
- Apache 1.3.x
- Omni HTTPd 2.0b1

Pasos Generales de Instalación

Los siguientes pasos deben realizarse en todas las instalaciones antes de las instrucciones específicas de cada servidor.

- Extraiga el archivo de distribución a un directorio de su elección. "C:\PHP3\" es un buen comienzo.
- Copie el archivo ’php3.ini-dist’ a su directorio ’%WINDOWS%’ y renómbrelo a ’php3.ini’. Su directorio ’%WINDOWS%’ es típicamente:
c:\windows para Windows 95/98
c:\winnt o c:\winnt40 para servidores NT
- Edite su archivo ’php3.ini’:
  - Necesitaá cambiar la opción ’extension_dir’ para que apunte a su php-install-dir, o a donde quiera que haya puesto sus archivos ’php3_*.dll’. P.ej.: c:\php3
  - Si está utilizando Omni Httpd, no siga el siguiente paso. Fije el ’doc_root’ para que apunte a la raiz web de sus servidores. P.ej.: c:\apache\htdocs o c:\webroot

- Elija qué módulos desearía cargar cuando comience el PHP. Puede descomentar las líneas: ’extension=php3_*.dll’ para cargar estos módulos. Algunos módulos requieren que tenga instaladas en sus sistema librerías adicionales para que el módulo funcione correctamente. El FAQ (http://www.php.net/FAQ.php) de PHP tiene más información sobre dónde obtener librerías de soporte. También puede cargar un módulo dinámicamente en su script utilizando: **dl("php_*.dll");**
- En el PWS y el IIS puede fijar el browscap.ini para que apunte a: ’c:\windows\system\inetsrv\browscap.ini’ bajo Windows 95/98 y a ’c:\winnt\system32\inetsrv\browscap.ini’ bajo NT Server.

Las DLL para las extensiones del PHP van precedidas de ’php3_’. Esto evita confusiones entre las extensiones del PHP y sus librerías de soporte.

Windows 95/98/NT y PWS/IIS 3

El método recomendado para configurar estos servidores es usar el archivo INF incluído con la distribución (php_iis_reg.inf). Quizás desee editar este archivo y asegurarse que las extensiones y directorios de instalación se ajustan a su configuración. O puede seguir los pasos que siguen para hacerlo de forma manual.

AVISO: Estos pasos conllevan el trabajar directamente con el registro de windows. Un error aquí puede dejar su sistema en un estado inestable. Le recomendamos encarecidamente que haga una copia de seguridad del registro con antelación. El equipo de Desarrollo del PHP no se hará responsable si se daña su registro.

- Ejecute Regedit.
- Navegue hasta: `HKEY_LOCAL_MACHINE /System /CurrentControlSet /Services /W3Svc /Parameters /ScriptMap`.
- En el menú de edición elija: `New->String Value`.
- Escriba la extensión que desea usar para sus script PHP. P.ej.: `.php3`
- Haga doble click en el nuevo valor de cadena y escriba la ruta al `php.exe` en el campo del valor. P.ej.: `c:\php3\php.exe %s %s`. La parte ’%s %s’ son MUY importantes, pues el PHP no funcionará correctamente sin ella.
- Repita estos pasos para cada extensión que desee asociar con los scripts PHP.
- Ahora navegue hasta: `HKEY_CLASSES_ROOT`
- En el menú de edición elija: `New->Key`.
- Déle a la clave el nombre de la extensión que preparó en la sección anterior. P.ej.: `.php3`
- Marque la nueva clave y en el panel del lado derecho haga doble click en "default value" y escriba `phpfile`.
- Repita el último paso para cada extensión que haya preparado en la sección previa.
- Ahora cree otra `New->Key` bajo `HKEY_CLASSES_ROOT` y denomínela `phpfile`.

- Marque la nueva clave `phpfile` y haga doble click en el panel derecho sobre "default value" y escriba `PHP Script`.
- Pulse el botón derecho sobre la clave `phpfile` y seleccione `New->Key` y llámela `Shell`.
- Pulse el botón derecho sobre la clave `Shell` y elija `New->Key` y llámela `open`.
- Pulse el botón derecho sobre la clave `open` y elija `New->Key` y llámela `command`.
- Marque la nueva clave `command` y en el panel derecho haga doble click sobre "default value" y entre la ruta hasta el `php.exe`. P.ej.: `c:\php3\php.exe -q %1`. (no olvide el `%1`).
- Salga del Regedit.

Los usuarios de PWS e IIS3 tienen ahora un sistema completamente operativo. Los usuarios del IIS3 también pueden usar una curiosa herramienta (http://www.genusa.com/iis/iiscfg.html) de Steven Genusa para configurar sus mapeados de script.

Windows NT e IIS 4

Para instalar el PHP en un NT Server con IIS 4, siga estas instrucciones:

- En el Controlador de Servicios de Internet (MMC), elija el sitio Web o el directorio de comienzo de una aplicación.
- Abra las propiedades del directorio (haciendo click derecho y eligiendo propiedades) y luego pulse sobre la pestaña Carpeta Inicial, Directorio Virtual o Directorio.
- Pulse el botón Configuración y luego pulse sobre la pestaña Mapas de Aplicación.
- Pulse en Añadir, y en la caja Programa, escriba: `c:\path-to-php-dir\php.exe %s %s`. DEBE mantene los %s %s al final, pues el PHP no funcionará correctamente si se equivoca al hacerlo.
- En la caja Extensión, escriba la extensión de fichero que desea asociar a los script de PHP. Debe repetir los pasos 5 y 6 para cada extensión que desee asociar con los scripts PHP (`.php3` y `.phtml` son habituales).
- Ajuste la seguridad apropiada (esto se realiza en el Controlador de Servicio de Internet (ISM)), y si su NT Server usa el sistema de archivos NTFS, añada derechos de ejecución para I_USR_ al directorio que contenga el `php.exe`.

Windows 9x/NT y Apache 1.3.x

Debe editar sus archivos `srm.conf` o `httpd.conf` para configurar el Apache para que trabaje con el ejecutable CGI del PHP.

Aunque puede haber algunas variaciones al configurar PHP bajo Apache, esta es lo suficientemente simple para ser usada por el novato. Por favor, consulte la Documentación del Apache para saber de las subsiguientes directivas de configuración.

- `ScriptAlias /php3/ "c:/ruta-al-dir-del-php/"`
- `AddType application/x-httpd-php3 .php3`
- `AddType application/x-httpd-php3 .phtml`
- `Action application/x-httpd-php3 "/php3/php.exe"`

Para utilizar la capacidad de marcado del código fuente, cree simplemente un script de PHP y pegue este código en él: `<?php show_source("script_original_php.php3"); ?>`. Sustituya `script_original_php.php3` por el nombre del archivo del que desea visualizar el código fuente (esta es la única forma de hacerlo). *Nota:* Bajo Win-Apache todas las barras invertidas de una ruta tal como: "c:\directory\file.ext", deben ser convertidas a barras hacia adelante.

Omni HTTPd 2.0b1 para Windows

Esta ha resultado ser la configuración más sencilla:

Paso 1: Instale el servidor Omni
Paso 2: Pulse el botón derecho sobre el icono azul del OmniHTTPd que está en la barrita del sistema y elija `Propiedades`
Paso 3: Pulse sobre `Web Server Global Settings`
Paso 4: En la pestaña 'External', escriba: `virtual = .php3 | actual = c:\ruta-al-dir-del-php\php.exe`
Paso 5: En la pestaña `Mime`, escriba: `virtual = wwwserver/stdcgi | actual = .php3`
Paso 6: Pulse en `OK`

Repita los pasos 2 a 6 para cada extensión que desee asociar al PHP.

Módulos del PHP

Tabla 3-1. Módulos del PHP

php3_calendar.dll	Funciones de conversión de calendario
php3_crypt.dll	Funciones de criptografía
php3_dbase.dll	Funciones para DBase
php3_dbm.dll	Emulación GDBM con la librería Berkeley DB2
php3_filepro.dll	Acceso SÓLO LECTURA a bases de datos filepro
php3_gd.dll	Funciones de librería GD para manipular GIF
php3_hyperwave.dll	Funciones de HyperWave
php3_imap4r2.dll	Funciones de IMAP 4
php3_ldap.dll	Funciones de LDAP
php3_msql1.dll	Cliente de mSQL 1
php3_msql2.dll	Cliente de mSQL 2
php3_mssql.dll	Cliente de MSSQL client (requiere las librerías de MSSQL DB
php3_mysql.dll	Funciones de MySQL
php3_nsmail.dll	Funciones de correo de Netscape

php3_oci73.dll	Funciones de Oracle
php3_snmp.dll	Funciones get y walk de SNMP (¡sólo en NT!)
php3_zlib.dll	Funciones de ZLib

¿Problemas?

Lea las PMF (FAQ)

Algunos problemas son más comunes que otros. Los más comunes están listados en las PMF (Preguntas Más Frecuentes) del PHP, que están en http://www.php.net/FAQ.php

Informes de error

Si cree que ha encontrado un error en el PHP, por favor infórmenos. Los desarrolladores del PHP probablemente no tengan conocimiento del mismo, y salvo si informa del mismo, pocas probabilidades habrá de que lo solucionen. Puede informar de los errores usando el sistema de rastreo de errores en http://bugs.php.net/.

Otros problemas

Si aún se encuentra atascado, alguien de la lista de correos del PHP puede ser capaz de ayudarle. Deberá buscar primero en los archivos, por si acaso alguien ya ha respondido a otra persona que tuvo el mismo problema que usted. Los archivos están disponibles desde la página de soporte en http://www.php.net/. Para suscribirse a la lista de correo de PHP, envíe un correo vacío a php-general-subscribe@lists.php.net (mailto:php-general-subscribe@lists.php.net). La dirección de la lista de correo es `php-general@lists.php.net`.

Si desea ayuda sobre la lista de correo, intente ser preciso y de los detalles necesarios sobre su entorno (qué sistema operativo, qué versión de PHP, qué servidor web, si está ejecutando el PHP como CGI o como módulo de servidor, etc.) y también código suficiente para que otros puedan reproducir y comprobar su problema.

Capítulo 4. Configuración

El archivo de configuración

El archivo de configuración (llamado `php3.ini` en PHP 3.0, y simplemente `php.ini` a partir del PHP 4.0) es leído cuando arranca el PHP. Para las versiones de PHP como módulo de servidor esto sólo ocurre una vez al arrancar el servidor web. Para la versión CGI, esto ocurre en cada llamada.

Cuando se utiliza PHP como módulo Apache, también puede cambiar los ajustes de configuración utilizando directivas en los archivos de configuración del Apache y en los .htaccess.

Con el PHP 3.0 hay directivas Apache que se corresponden a cada uno de los ajustes de configuración del `php3.ini`, con la excepción que su nombre va precedido de "php3_".

Con el PHP 4.0 sólo hay unas pocas directivas de Apache que le permiten cambiar los ajustes de configuración del PHP.

php_value *`nombre`* *`valor`*

Fija el valor de la variable especificada.

php_flag *`nombre`* *`on|off`*

Fija una opción de configuración de tipo Boolean.

php_admin_value *`nombre`* *`valor`*

Fija el valor de la variable especificada. Los ajustes de configuración de tipo "Admin" sólo se pueden fijar desde los archivos principales de configuración del Apache, y no desde los .htaccess.

php_admin_flag *`nombre`* *`on|off`*

Fija una opción de configuración de tipo Boolean.

Puede ver los ajustes de los valores de configuración en la salida de phpinfo(). También puede acceder a los valores individuales de los ajustes de configuración utilizando get_cfg_var().

Directivas Generales de Configuración

`asp_tags` boolean

Permite el uso de las etiquetas estilo ASP <% %> además de las habituales etiquetas <?php ?>. También se incluye el atajo para imprimir variables <%= $valor %>. Para más información, vea Escapando del HTML.

> **Nota:** El soporte para etiquetas estilo ASP se añadió en la 3.0.4.

`auto_append_file` string

Especifica el nombre de un archivo que es troceado automáticamente después del archivo principal. El archivo se incluye como si fuese llamado mediante la función include(), así que se utiliza include_path.

El valor especial none desconecta la adición automática de archivos.

> **Nota:** Si el script es terminado con exit(), *no* tendrá lugar la adición automática.

`auto_prepend_file` string

Especifica el nombre de un archivo que es troceado automáticamente antes del archivo principal. Specifies the name of a file that is automatically parsed before the main file. El archivo se incluye como si fuese llamado mediante la función include(), así que se utiliza include_path.

El valor especial none desconecta la adición automática de archivos.

`cgi_ext` string

`display_errors` boolean

Determina si los errores se visualizan en pantalla como parte de la salida en HTML o no.

`doc_root` string

"Directorio raiz" del PHP en el servidor. Sólo se usa si no está vacío. Si el PHP se configura con safe mode, no se sirven archivos fuera de este directorio.

`engine` boolean

Esta directiva sólo es realmente útil en la versión de PHP como módulo Apache. Se utiliza por sitios que desean habilitar la ejecución del PHP directorio por directorio o en base a cada servidor virtual. Poniendo **`php3_engine off`** en los sitios apropiados del archivo `httpd.conf`, se puede habilitar o deshabilitar el PHP.

`error_log` string

Nombre del fichero para registrar los errores de un script. Si se utiliza el valor especial `syslog`, los errores se envían al registro de errores del sistema. En UNIX se refiere a syslog(3) y en Windows NT al registro de eventos. El registro de errores del sistema no es soportado bajo Windows 95.

`error_reporting` integer

Fija el nivel de informe de errores. El parámetro es un entero que representa un campo de bits. Sume los valores de los niveles de informe de error que desea.

Tabla 4-1. Niveles de Informe de Errores

valor de bit	informe habilitado
1	errores normales
2	avisos normales
4	errores del troceador (parser)
8	avisos de estilo no críticos

El valor por defecto para esta directiva es 7 (se muestran los errores normales, avisos normales y errores de parser).

`open_basedir` string

Limita los archivos que se pueden abrir por el PHP al árbol de directorios especificado.

Cuando un script intenta abrir un archivo con, por ejemplo, fopen o gzopen, se comprueba su localización. Si el fichero está fuera del árbol de directorios especificado, PHP se negará a abrirlo. Todos los enlaces simbólicos son resueltos, de modo que no es posible evitar esta limitación usando uno de ellos.

El valor especial . indica que el directorio base será aquel en el que reside el script.

Bajo Windows, separe los directorios mediante punto y coma. En el resto de sistemas, sepárelos con dos puntos ":". Como módulo de Apache, los senderos para open_basedir de los directorios padre se heredan ahora automáticamente.

> **Nota:** El soporte para directorios múltiples se añadió en la 3.0.7.

El valor por defecto es permitir abrir todos los archivos.

`gpc_order` string

Fija el order de troceo de variables GET/POST/COOKIE. El valor por defecto de esta directiva es "GPC". Fijándola, por ejemplo, a "GP", hará que el PHP ignore por completo las cookies y que sobreescriba las variables recibidas por GET con las que tengan el mismo nombre y vengan por POST.

`ignore_user_abort` string

Por defecto está a on. Si se cambia a off, los script terminarán tan pronto como intenten enviar algo después de que un cliente ha roto la conexión. ignore_user_abort().

`include_path` string

Especifica una lista de directorios en los que las funciones require(), include() y **fopen_with_path()** buscan los archivos. El formato es similar a la variable de entorno de sistema

PATH: una lista de directorios separados por dos puntos en UNIX o por punto y coma en Windows.

Ejemplo 4-1. include_path en UNIX

```
include_path=.:/home/httpd/php-lib
```

Ejemplo 4-2. include_path en Windows

```
include_path=".;c:\www\phplib"
```

El valor por defecto para esta directiva es `.` (sólo el directorio actual).

`isapi_ext` string

`log_errors` boolean

Dice si los mensajes de error de los script deben ser registrados o no en el registro del servidor. Esta opción, por tanto, es específica del mismo.

`magic_quotes_gpc` boolean

Fija el estado magic_quotes para operaciones GPC (Get/Post/Cookie). Si magic_quotes vale on, todas las ’ (comilla sencilla), " (comilla doble), \ (barra invertida) y los NUL son automáticamente marcados con una barra invertida. Si además magic_quotes_sybase vale on, la comilla sencilla es marcada con otra comilla sencilla en lugar de la barra invertida.

`magic_quotes_runtime` boolean

Si se habilita *`magic_quotes_runtime`*, muchas de las funciones que devuelven datos de algún tipo de fuente externa incluyendo bases de datos y archivos de texto devolverán las comillas marcadas con una barra invertida. Si también está activo *`magic_quotes_sybase`*, la comilla simple es marcada con una comilla simple en lugar de la barra invertida.

`magic_quotes_sybase` boolean

Si *`magic_quotes_sybase`* está a on, la comilla simple es marcada con una comilla simple en lugar de la barra invertida cuando están habilitados *`magic_quotes_gpc`* o *`magic_quotes_runtime`*.

`max_execution_time` integer

Fija el tiempo máximo en segundos que se le permite usar a un script antes de ser finalizado por el intérprete. Así se evita que scripts mal escritos puedan bloquear el servidor.

`memory_limit` integer

Fija el tamaño máximo de memoria en bytes que se permite reclamar a un script. Así se evita que script mal escritos se coman toda la memoria dispomible de un servidor.

`nsapi_ext` string

short_open_tag boolean

Indica si se debe permitir el formato corto (**<? ?>**) de la etiqueta de apertura del PHP. Si desea utilizar PHP en combinación con XML, deberá desactivar esta opción. Si está desactivada, deberá utilizar el formato largo de la etiqueta de apertura (**<?php ?>**).

sql.safe_mode boolean

track_errors boolean

Si está habilitada, el último mensaje de error estará siempre presente en la variable global $php_errormsg.

track_vars boolean

Si está activada, la información de entrada de GET, POST y de las cookies se puede encontrar en las matrices asociativas $HTTP_GET_VARS,$HTTP_POST_VARS y $HTTP_COOKIE_VARS respectivamente.

upload_tmp_dir string

El directorio temporal utilizado para almacenar archivos cuando se envían al servidor. Debe tener permiso de escritura para el usuario bajo el que corra el PHP.

user_dir string

El nombre base del directorio utilizado bajo el directorio inicial de un usuario para los archivos PHP. Por ejemplo: `paginas_html`.

warn_plus_overloading boolean

Si está activada, esta opción hace que el PHP muestre un aviso cuando el operador suma (`+`) se utiliza en cadenas. Así es más fácil encontrar scripts que necesitan ser reescritos utilizando en su lugar el concatenador de cadenas (`.`).

Directivas de Configuración de Correo

SMTP string

Nombre DNS o dirección IP del servidor de SMTP que el PHP bajo Windows deberá usar para enviar correo con la función mail().

sendmail_from string

La dirección del remitente (`"De:"`) para los correos enviados desde PHP bajo Windows.

sendmail_path string

Localización del programa **sendmail**. Generalmente `/usr/sbin/sendmail` o `/usr/lib/sendmail`. **configure** intenta localizarle este archivo lo mejor que puede y fijar un valor por defecto, pero en caso de fallo, lo puede usted fijar aquí.

Los sistemas que no usan sendmail deberán fijar esta directiva al nombre del programa alternativo que ofrezca su sistema de correo, si es que existe. Por ejemplo, los usuarios del Qmail (http://www.qmail.org/) pueden fijarlo normalmente a `/var/qmail/bin/sendmail`.

Directivas de Configuración de Modo Seguro

`safe_mode` boolean

Para activar el modo seguro del PHP. Lea el Capítulo de seguridad para más información.

`safe_mode_exec_dir` string

Si el PHP se utiliza en modo seguro, la función system() y el resto de funciones que ejecutan programas del sistema se niegan a ejecutar programas que no estén en este directorio.

Directivas de Configuración del Debugger

`debugger.host` string

Nombre DNS o dirección IP del servidor usado por el debugger.

`debugger.port` string

Número de puerto usado por el debugger.

`debugger.enabled` boolean

Indica si el debugger está habilitado o no.

Directivas de Carga de Extensiones

`enable_dl` boolean

Esta directiva sólo es útil en la versión del PHP como módulo del Apache. Puede habilitar o deshabilitar para un servidor virtual o para un directorio la carga dinámica de extensiones de PHP mediante dl().

La razón principal para deshabilitar la carga dinámica es la seguridad. Con la carga dinámica es posible ignorar las restricciones safe_mode y open_basedir.

El valor por defecto es permitir la carga dinámica, excepto cuando se usa el modo seguro. En modo seguro, siempre es imposible usar dl().

`extension_dir` string

En qué directorio debe buscar el PHP las extensiones cargables dinámicamente.

extension string

Qué extensiones dinámicas debe cargar el PHP cuando arranca.

Directivas de Configuración de MySQL

mysql.allow_persistent boolean

Si permitir o no conexiones MySQL persistentes.

mysql.default_host string

El servidor por defecto para utilizar cuando se conecte al servidor de bases de datos si no se especifica otro distinto.

mysql.default_user string

El nombre de usuario por defecto para utilizar cuando se conecta al servidor de base de datos si no se especifica otro.

mysql.default_password string

La clave por defecto para utilizar cuando se conecta al servidor de base de datos si no se especifica otro.

mysql.max_persistent integer

El número máximo de conexiones persistentes de MySQL por proceso.

mysql.max_links integer

El número máximo de conexiones de MySQL por proceso, incluyendo las persistentes.

Directivas de Configuración de mSQL

msql.allow_persistent boolean

Si se permiten o no conexiones persistentes de mSQL.

msql.max_persistent integer

El número máximo de conexiones persistentes mSQL por proceso.

msql.max_links integer

El número máximo de conexiones de mSQL por proceso, incluyendo las persistentes.

Directivas de Configuración de Postgres

pgsql.allow_persistent boolean

Si se permiten o no conexiones persistentes de Postgres.

pgsql.max_persistent integer

El número máximo de conexiones persistentes Postgres por proceso.

pgsql.max_links integer

El número máximo de conexiones de Postgres por proceso, incluyendo las persistentes.

SESAM Configuration Directives

sesam_oml string

Name of BS2000 PLAM library containing the loadable SESAM driver modules. Required for using SESAM functions. The BS2000 PLAM library must be set ACCESS=READ,SHARE=YES because it must be readable by the apache server's user id.

sesam_configfile string

Name of SESAM application configuration file. Required for using SESAM functions. The BS2000 file must be readable by the apache server's user id.

The application configuration file will usually contain a configuration like (see SESAM reference manual):

```
CNF=B
NAM=K
NOTYPE
```

sesam_messagecatalog string

Name of SESAM message catalog file. In most cases, this directive is not neccessary. Only if the SESAM message file is not installed in the system's BS2000 message file table, it can be set with this directive.

The message catalog must be set ACCESS=READ,SHARE=YES because it must be readable by the apache server's user id.

Directivas de Configuración de Sybase

sybase.allow_persistent boolean

Si se permiten o no conexiones persistentes de Sybase.

sybase.max_persistent integer

El número máximo de conexiones persistentes Sybase por proceso.

`sybase.max_links` integer

El número máximo de conexiones de Sybase por proceso, incluyendo las persistentes.

Directivas de Configuración de Sybase-CT

`sybct.allow_persistent` boolean

Si se permiten o no conexiones persistentes de Sybase-CT. El valor por defecto es on.

`sybct.max_persistent` integer

El número máximo de conexiones persistentes Sybase-CT por proceso. El valor por defecto es -1, que significa ilimitadas.

`sybct.max_links` integer

El número máximo de conexiones de Sybase-CT por proceso, incluyendo las persistentes. El valor por defecto es -1, que significa ilimitadas.

`sybct.min_server_severity` integer

Los mensajes de servidor con gravedad mayor o igual que `sybct.min_server_severity` serán reportados como avisos. Este valor también se puede cambiar desde un script usando la función sybase_min_server_severity(). El valor por defecto es 10, que reporta los errores de información con gravedad o mayores.

`sybct.min_client_severity` integer

Los mensajes de librería de cliente con gravedad mayor o igual que `sybct.min_client_severity` serán reportados como avisos. Este valor también se puede cambiar desde un script usando la función sybase_min_client_severity(). El valor por defecto es 10, que desconecta los avisos.

`sybct.login_timeout` integer

El número máximo de segundos de espera por un intento de conexión con éxito antes de indicar un fallo. Nótese que si se ha excedido `max_execution_time` cuando finaliza la espera de un intento de conexión, el script será finalizado antes de que se pueda tomar una acción en caso de fallo. El valor por defecto es 1 minuto.

`sybct.timeout` integer

El número máximo de segundos de espera por una operación de consulta o select_db con éxito antes de indicar un fallo. Nótese que si se ha excedido `max_execution_time` cuando finaliza la espera de un intento de conexión, el script será finalizado antes de que se pueda tomar una acción en caso de fallo. El valor por defecto es sin límite.

`sybct.hostname` string

El nombre de la máquina desde la que dice estarse conectando, para que se visualize con **sp_who()**. El valor por defecto es "none".

Directivas de Configuración de Informix

ifx.allow_persistent boolean

Si se permiten o no conexiones persistentes de Informix.

ifx.max_persistent integer

El número máximo de conexiones persistentes de Informix por proceso.

ifx.max_links integer

El número máximo de conexiones Informix por proceso, incluyendo las persistentes.

ifx.default_host string

El servidor por defecto al que conectarse si no se especifica uno en ifx_connect() o en ifx_pconnect().

ifx.default_user string

El id de usuario por defecto para utilizar si no se especifica uno en ifx_connect() o en ifx_pconnect().

ifx.default_password string

La clave por defecto para utilizar si no se especifica uno en ifx_connect() o en ifx_pconnect().

ifx.blobinfile boolean

Fíjelo a `TRUE` si desea recibir las columnas blob (objetos binarios grandes) en un archivo, y a `FALSE` si las desea en memoria. Puede cambiar el ajuste en tiempo de ejecución utilizando ifx_blobinfile_mode().

ifx.textasvarchar boolean

Fíjelo a `TRUE` si desea recibir las columnas TEXT como cadenas normales en las instrucciones select, y a `FALSE` si quiere usar parámetros de identificador de blobs. Puede cambiar el ajuste en tiempo de ejecución utilizando ifx_textasvarchar().

ifx.byteasvarchar boolean

Fíjelo a `TRUE` si desea devolver las columnas BYTE como cadenas normales en las instrucciones select, y a `FALSE` si quiere usar parámetros de identificador de blobs. Puede cambiar el ajuste en tiempo de ejecución utilizando ifx_byteasvarchar().

ifx.charasvarchar boolean

Fíjelo a `TRUE` si desea suprimir los espacios a la derecha de las columnas CHAR cuando las solicita.

ifx.nullformat boolean

Fíjelo a `TRUE` si desea que las columnas `NULL` (nulas) se devuelvan como la cadena literal "`NULL`", y a `FALSE` si desea que se devuelvan como la cadena vacía "". Puede cambiar el ajuste en tiempo de ejecución utilizando ifx_nullformat().

Directivas de Configuración de Matemática BC

bcmath.scale integer

Número de dígitos decimales para todas las funciones de bcmath.

Directivas de Configuración de Capacidades de los Navegadores

browscap string

Nombre del archivo de capacidades del navegador. Vea también get_browser().

Directivas Unificadas de Configuración de ODBC

uodbc.default_db string

Fuentes de datos ODBC a utilizar si no se especifica una en odbc_connect() o en odbc_pconnect().

uodbc.default_user string

Nombre de usuario si no se especifica uno en odbc_connect() o en odbc_pconnect().

uodbc.default_pw string

Clave para usar si no se especifica una en odbc_connect() o en odbc_pconnect().

uodbc.allow_persistent boolean

Si se permiten o no conexiones persistentes de ODBC.

uodbc.max_persistent integer

El número máximo de conexiones persistentes de ODBC por proceso.

uodbc.max_links integer

El número máximo de conexiones ODBC por proceso, incluyendo las persistentes.

Capítulo 5. Seguridad

PHP es un potente lenguaje y el interprete, tanto incluido en el servidor web como modulo o ejecutado como un binario CGI, puede acceder a ficheros, ejecutar comandos y abrir comunicaciones de red en el servidor. Todas estas caracteristicas hacen que lo que se ejecute en el servidor web sea inseguro por defecto. PHP ha sido disenado especificamente, para ser un lenguaje mas seguro para escribir programas CGI, que Perl o C y con la correcta seleccion de las opciones de configuración del tiempo de compilación y ejecucion se consigue la exacta combinación de libertad y seguridad que se necesita.

Ya que existen diferentes modos de utilizar PHP, existen multitud de opciones de configuración que permiten controlar su funcionamiento. Una gran selección de opciones garantiza que se pueda usar PHP para diferentes usos, pero tambien significa que existen combinaciones de estas opciones y configuraciones del servidor que producen instalaciones inseguras. Este capitulo explica las diferentes combinaciones de opciones de configuración y las situaciones donde pueden ser usadas de manera segura.

Binarios CGI

Posibles ataques

Usando PHP como un binario CGI es una opción para instalaciones que por cualquier causa no quieren integrar PHP como modulo en el software servidor (p.ej: Apache), o usaran PHP con diferentes clases de CGI wrappers para crear entornos chroot y setuid seguros para los scripts. Esta configuración implica generalmente el instalar el binario ejecutable de PHP en el directorio cgi-bin del servidor web. El documento del CERT CA-96.11 (http://www.cert.org/advisories/CA-96.11.interpreters_in_cgi_bin_dir.html) recomienda no instalar interpretes en cgi-bin. Aunque el binario PHP puede ser usado como interprete independiente, PHP esta diseñado para prevenir los ataques que esta configuración hace posible.

- Accediendo a ficheros del sistema: `http://my.host/cgi-bin/php?/etc/passwd`

  La información introducida despues del signo de interrogación (?) es transferida como argumento de la línea de comando al intérprete por el interfaz del CGI. Normalmente los interpretes abren y ejecutan el fichero especificado como el primer argumento en la línea de comando.

  Cuando se ejecuta como un CGI script, PHP rechaza interpretar los argumentos de la línea de comando.

- Accediendo cualquier documento web en el servidor: `http://my.host/cgi-bin/php/secret/doc.html`

  La información con el camino (path) de la URL despues del nombre del binario PHP, `/secret/doc.html` es usada convencionalmente para especificar el nombre del fichero que sera abierto e interpretado por el programa CGI. Normalmente, algunas directivas del servidor web (Apache:Action) son usadas para redireccionar peticiones de documentos como `http://my.host/secret/script.php3` al interprete PHP. Con esta configuración, el servidor web comprueba primero los permisos de acceso al directorio `/secret`, y despues crea la petición redireccionada `http://my.host/cgi-bin/php/secret/script.php3`. Desafortunadamente, si la petición es hecha de esta forma en un principio, el servidor web no comprueba los permisos de acceso del fichero `/secret/script.php3`, sino solamente del fichero `/cgi-bin/php`. De esta

manera cualquier usuario que pueda acceder `/cgi-bin/php` tambien puede acceder a cualquier documento protegido en el servidor web.

En PHP, a la hora de compilar, la opción de configuracion --enable-force-cgi-redirect y las directivas de configuracion a la hora de ejecutar doc_root y user_dir pueden ser usadas para prevenir este ataque, si el arbol de documentos del servidor tiene cualquier directorio con acceso restringido. Ver mas adelante la explicacion de las diferentes combinaciones.

Caso 1: solamente se sirven ficheros publicos

Si tu servidor no contiene informacion que este protegida con clave o acceso de control de IPs, no se necesitan estas opciones de configuracion. Si tu servidor web no permite realizar redireccionamientos, o el servidor no tiene modo de comunicar al binario PHP que la peticion es una peticion segura redireccionada, podeis especificar la opcion --disable-force-cgi-redirect en el script de configuracion. De todas maneras, teneis que aseguraros que vuestros scripts PHP no confíen en la manera al llamar al script, ni de forma directa `http://my.host/cgi-bin/php/dir/script.php3` o por redireccion `http://my.host/dir/script.php3`.

Redireccionamiento puede ser configurado en Apache usando las directivas AddHandler y Action (ver mas abajo).

Caso 2: usando --enable-force-cgi-redirect

Esta opcion a la hora de compilar previene que alguien llame PHP directamente con una url como la siguiente `http://my.host/cgi-bin/php/secretdir/script.php3`. PHP solamente analizara en este modo si ha pasado por una regla de redireccionamiento en el servidor.

Normalmente la redireccion en la configuracion de Apache es hecha con la siguientes directivas:

```
Action php3-script /cgi-bin/php
AddHandler php3-script .php3
```

Esta opcion ha sido solo comprobada con el servidor web Apache, y depende de Apache para fijar la variable de entorno CGI no estandar REDIRECT_STATUS en las peticiones de redireccionamiento. Si tu servidor web no soporta ningun modo para informar si una peticion es directa o redireccionada, no podeis usar esta opcion y debereis usar alguno de los otros modos de ejecucion de la version CGI documentados aqui.

Caso 3: Usando doc_root or user_dir

Incluir contenidos activos, como script y ejecutables, en el directorio de documentos del servidor web, es algunas veces considerada una practica insegura. Si por algun fallo de configuracion, los scripts no son ejecutados pero mostrados como documentos HTML, cualquiera podra conseguir codigo registrado o informacion de seguridad, como p.ej: claves de acceso. Por ello, muchos administradores prefieren utilizar otra estructura de directorios que contenga solamente los scripts, los cuales seran solamente accesibles via PHP CGI, y por ello siempre seran interpretados y no mostrados.

Habra que tener en cuenta que si el metodo que asegura que las peticiones no son redireccionadas, como hemos descrito en la seccion anterior, no esta disponible, sera necesario configurar un script doc_root que sea diferente del "web document root".

Podeis definir el script PHP "document root" con la directiva de configuracion doc_root en el fichero de configuracion, o definir la variable de entorno PHP_DOCUMENT_ROOT. Si esta definida, la version CGI de PHP siempre obtendra el nombre del fichero a abrir con *`doc_root`* y el camino (path) utilizado en la peticion, asi podeis estar seguros que ningun script sera ejecutado fuera de este directorio (excepto para *`user_dir`*, ver a continuacion)

Otra opcion que se puede usar aqui es user_dir. Cuando user_dir no esta definido, lo unico que controla la apertura del fichero es *`doc_root`*. Si intentamos abrir una url tal como esta `http://my.host/~user/doc.php3` no se abrira un fichero en el directorio de usuarios, en su lugar se abrira un fichero llamado `~user/doc.php3` en el directorio doc_root. (si, un directorio que empieza por tilde [~]).

Si user_dir esta definido por ejemplo como `public_php`, una peticion tal como `http://my.host/~user/doc.php3`, abrira un fichero llamado `doc.php3` en el directorio llamado `public_php` del directorio "home" del usuario. Si el directorio del usuario es `/home/user`, el fichero ejecutado sera `/home/user/public_php/doc.php3`.

La expansion de *`user_dir`* ocurre sin tener en cuenta la configuracion de *`doc_root`*, de este modo se puede controlar los accesos al directorio principal (document root) y al directorio de usuario separadamente.

Caso 4: Analizador PHP fuera del arbol web.

Una opcion muy segura es poner el analizador binario PHP, en algun lugar fuera del arbol de ficheros web. Por ejemplo en `/usr/local/bin`. La unica pega real de esta opcion es que habra que poner una linea similar a:

```
#!/usr/local/bin/php
```

como primera linea en cualquier fichero que contenga codigo PHP. Tambien sera necesario asignar al fichero permisos de ejecucion. De esta manera, es tratado de la misma manera que cualquier otro CGI script escrito en Perl o sh o otro lenguaje utilizado para scripts y que utilicen el mecanismo `#!` para ejecutarse.

Para conseguir que PHP maneje correctamente con esta configuracion, la informacion de PATH_INFO y PATH_TRANSLATED, el analizador PHP deberia ser compilado con la opcion de configuracion --enable-discard-path.

Modulo Apache

Cuando PHP es usado como modulo Apache, hereda los permisos de usuario de Apache (normalmente "nobody")

Parte II. Referencia del Lenguaje

Capítulo 6. Síntaxis básica

Saliendo de HTML

Para interpretar un archivo, php símplemente interpreta el texto del archivo hasta que encuentra uno de los carácteres especiales que delimitan el inicio de código PHP. El intérprete ejecuta entonces todo el código que encuentra, hasta que encuentra una etiqueta de fin de código, que le dice al intérprete que siga ignorando el código siguiente. Este mecanismo permite embeber código PHP dentro de HTML: todo lo que está fuera de las etiquetas PHP se deja tal como está, mientras que el resto se interpreta como código.

Hay cuatro conjuntos de etiquetas que pueden ser usadas para denotar bloques de código PHP. De estas cuatro, sólo 2 (<?php. . .?> y <script language="php">. . .</script>) están siempre disponibles; el resto pueden ser configuradas en el fichero de `php.ini` para ser o no aceptadas por el intérprete. Mientras que el formato corto de etiquetas (short-form tags) y el estilo ASP (ASP-style tags) pueden ser convenientes, no son portables como la versión de formato largo de etiquetas. Además, si se pretende embeber código PHP en XML o XHTML, será obligatorio el uso del formato <?php. . .?> para la compatibilidad con XML.

Las etiquetas soportadas por PHP son:

Ejemplo 6-1. Formas de escapar de HTML

```
1.  <?php echo("si quieres servir documentos XHTML o XML, haz como aqu&iacute;\n"); ?>

2.  <? echo ("esta es la m&aacute;s simple, una instrucci&oacute;n de procesado SGML \n"); ?
    <?= expression ?> Esto es una abreviatura de "<? echo expression ?>"

3.  <script language="php">
        echo ("muchos editores (como FrontPage) no
              aceptan instrucciones de procesado");
    </script>

4.  <% echo ("Opcionalmente, puedes usar las etiquetas ASP"); %>
    <%= $variable; # Esto es una abreviatura de "<% echo . . ." %>
```

El método primero, <?php. . .?>, es el más conveniente, ya que permite el uso de PHP en código XML como XHTML.

El método segundo no siempre está disponible. El formato corto de etiquetas está disponible con la función **short_tags()** (sólo PHP 3), activando el parámetro del fichero de configuración de PHP short_open_tag, o compilando PHP con la opción --enable-short-tags del comando **configure**. Aunque esté activa por defecto en php.ini-dist, se desaconseja el uso del formato de etiquetas corto.

El método cuarto sólo está disponible si se han activado las etiquetas ASP en el fichero de configuración: asp_tags.

> **Nota:** El soporte de etiquetas ASP se añadió en la versión 3.0.4.

Nota: No se debe usar el formato corto de etiquetas cuando se desarrollen aplicaciones o librerías con intención de redistribuirlas, o cuando se desarrolle para servidores que no están bajo nuestro control, porque puede ser que el formato corto de etiquetas no esté soportado en el servidor. Para generar código portable y redistribuíble, asegúrate de no usar el formato corto de etiquetas.

La etiqueta de fin de bloque incluirá tras ella la siguiente línea si hay alguna presente. Además, la etiqueta de fin de bloque lleva implícito el punto y coma; no necesitas por lo tanto añadir el punto y coma final de la última línea del bloque PHP.

PHP permite estructurar bloques como:

Ejemplo 6-2. Métodos avanzados de escape

```
<?php
if ($expression) {
    ?>
    <strong>This is true.</strong>
    <?php
} else {
    ?>
    <strong>This is false.</strong>
    <?php
}
?>
```

Este ejemplo realiza lo esperado, ya que cuando PHP encuentra las etiquetas ?> de fin de bloque, empieza a escribir lo que encuentra tal cual hasta que encuentra otra etiqueta de inicio de bloque. El ejemplo anterior es, por supuesto, inventado. Para escribir bloques grandes de texto generamente es más eficiente separalos del código PHP que enviar todo el texto mediante las funciones echo(), print() o similares.

Separación de instrucciones

Las separación de instrucciones se hace de la misma manera que en C o Perl - terminando cada declaración con un punto y coma.

La etiqueta de fin de bloque (?>) implica el fin de la declaración, por lo tanto lo siguiente es equivalente:

```
<?php
    echo "This is a test";
?>

<?php echo "This is a test" ?>
```

Comentarios

PHP soporta el estilo de comentarios de 'C', 'C++' y de la interfaz de comandos de Unix. Por ejemplo:

```
<?php
    echo "This is a test"; // This is a one-line c++ style comment
    /* This is a multi line comment
       yet another line of comment */
    echo "This is yet another test";
    echo "One Final Test"; # This is shell-style style comment
?>
```

Los estilos de comentarios de una linea actualmente sólo comentan hasta el final de la linea o el bloque actual de código PHP, lo primero que ocurra.

```
<h1>This is an <?php # echo "simple";?> example.</h1>
<p>The header above will say 'This is an example'.
```

Hay que tener cuidado con no anidar comentarios de estilo 'C', algo que puede ocurrir al comentar bloques largos de código.

```
<?php
 /*
    echo "This is a test"; /* This comment will cause a problem */
 */
?>
```

Los estilos de comentarios de una linea actualmente sólo comentan hasta el final de la linea o del bloque actual de código PHP, lo primero que ocurra. Esto implica que el código HTML tras // ?> seráa impreso: ?> sale del modo PHP, retornando al modo HTML, el comentario // no le influye.

Capítulo 7. Types

PHP soporta los siguientes tipos:

- array
- números en punto flotante
- entero
- objeto
- cadena

El tipo de una variable normalmente no lo indica el programador; en su lugar, lo decide PHP en tiempo de ejecución dependiendo del contexto en el que se utilice esa variable.

Si se quisiese obligar a que una variable se convierta a un tipo concreto, se podría forzar la variable o usar la función settype() para ello.

Nótese que una variable se puede comportar de formas diferentes en ciertas situaciones, dependiendo de qué tipo sea en ese momento. Para más información, vea la sección Conversión de Tipos.

Enteros

Los enteros se puede especificar usando una de las siguientes sintaxis:

```
$a = 1234; # número decimal
$a = -123; # un número negativo
$a = 0123; # número octal (equivalente al 83 decimal)
$a = 0x12; # número hexadecimal (equivalente al 18 decimal)
```

Números en punto flotante

Los números en punto flotante ("double") se pueden especificar utilizando cualquiera de las siguientes sintaxis:

```
$a = 1.234; $a = 1.2e3;
```

Cadenas

Las cadenas de caracteres se pueden especificar usando uno de dos tipos de delimitadores.

Si la cadena está encerrada entre dobles comillas ("), las variables que estén dentro de la cadena serán expandidas (sujetas a ciertas limitaciones de interpretación). Como en C y en Perl, el carácter de barra invertida ("\") se puede usar para especificar caracteres especiales:

Tabla 7-1. Caracteres protegidos

secuencia	significado
`\n`	Nueva línea
`\r`	Retorno de carro
`\t`	Tabulación horizontal
`\\`	Barra invertida
`\$`	Signo del dólar
`\"`	Comillas dobles
`\[0-7]{1,3}`	la secuencia de caracteres que coincida con la expresión regular es un carácter en notación octal
`\x[0-9A-Fa-f]{1,2}`	la secuencia de caracteres que coincida con la expresión regular es un carácter en notación hexadecimal

Se puede proteger cualquier otro carácter, pero se producirá una advertencia en el nivel de depuración más alto.

La segunda forma de delimitar una cadena de caracteres usa el carácter de comilla simple ("’"). Cuando una cadena va encerrada entre comillas simples, los únicos caracteres de escape que serán comprendidos son "\\" y "\’". Esto es por convenio, así que se pueden tener comillas simples y barras invertidas en una cadena entre comillas simples. Las variables *no* se expandirán dentro de una cadena entre comillas simples.

Otra forma de delimitar cadenas es usando la sintaxis de documento incrustado ("<<<"). Se debe proporcionar un identificador después de <<<, después la cadena, y después el mismo identificador para cerrar el entrecomillado.

Ejemplo 7-1. He aquí un ejemplo de entrecomillado de cadenas con sintaxis de documento incrustado

```
$str = <<<EOD
Ejemplo de cadena
Expandiendo múltiples líneas
usando sintaxis de documento incrustado.
EOD;
```

> **Nota:** La sintaxis de documento incristado fue añadida en PHP 4.

Las cadenas se pueden concatenar usando el operador ’.’ (punto). Nótese que el operador ’+’ (suma) no sirve para esto. Por favor mire Operadores de cadena para más información.

Se puede acceder a los caracteres dentro de una cadena tratándola como un array de caracteres indexado numéricamente, usando una sintaxis similar a la de C. Vea un ejemplo más abajo.

Ejemplo 7-2. Algumos ejemplos de cadenas

```
<?php
/* Asignando una cadena. */
$str = "Esto es una cadena";

/* Añadiendo a la cadena. */
$str = $str . " con algo más de texto";

/* Otra forma de añadir, incluye un carácter de nueva línea protegido. */
$str .= " Y un carácter de nueva línea al final.\n";

/* Esta cadena terminará siendo ’<p>Número: 9</p>’ */
$num = 9;
$str = "<p>Número: $num</p>";

/* Esta será ’<p>Número: $num</p>’ */
$num = 9;
$str = ’<p>Número: $num</p>’;

/* Obtener el primer carácter de una cadena  */
$str = ’Esto es una prueba.’;
$first = $str[0];

/* Obtener el último carácter de una cadena. */
$str = ’Esto es aún una prueba.’;
$last = $str[strlen($str)-1];
?>
```

Conversión de cadenas

Cuando una cadena se evalúa como un valor numérico, el valor resultante y el tipo se determinan como sigue.

La cadena se evaluará como un doble si contiene cualquiera de los caracteres ’.’, ’e’, o ’E’. En caso contrario, se evaluará como un entero.

El valor viene dado por la porción inicial de la cadena. Si la cadena comienza con datos de valor numérico, este será el valor usado. En caso contrario, el valor será 0 (cero). Los datos numéricos válidos son un signo opcional, seguido por uno o más dígitos (que opcionalmente contengan un punto decimal), seguidos por un exponente opcional. El exponente es una ’e’ o una ’E’ seguidos por uno o más dígitos.

Cuando la primera expresión es una cadena, el tipo de la variable dependerá de la segunda expresión.

```
$foo = 1 + "10.5";              // $foo es doble (11.5)
$foo = 1 + "-1.3e3";            // $foo es doble (-1299)
$foo = 1 + "bob-1.3e3";         // $foo es entero (1)
$foo = 1 + "bob3";              // $foo es entero (1)
$foo = 1 + "10 Cerditos";     // $foo es entero (11)
$foo = 1 + "10 Cerditos"; // $foo es entero (11)
```

```
$foo = "10.0 cerdos " + 1;        // $foo es entero (11)
$foo = "10.0 cerdos " + 1.0;      // $foo es double (11)
```

Para más información sobre esta conversión, mire en la página del manual de Unix strtod(3).

Si quisiera probar cualquiera de los ejemplos de esta sección, puede cortar y pegar los ejemplos e insertar la siguiente línea para ver por sí mismo lo que va ocurriendo:

```
echo "\$foo==$foo; el tipo es " . gettype( $foo ) . "<br>\n";
```

Arrays

Los arrays actualmente actúan tanto como tablas hash (arrays asociativos) como arrays indexados (vectores).

Arrays unidimensionales

PHP soporta tanto arrays escalares como asociativos. De hecho, no hay diferencias entre los dos. Se puede crear una array usando las funciones list() o array(), o se puede asignar el valor de cada elemento del array de manera explícita.

```
$a[0] = "abc";
$a[1] = "def";
$b["foo"] = 13;
```

También se puede crear un array simplemente añadiendo valores al array. Cuando se asigna un valor a una variable array usando corchetes vacíos, el valor se añadirá al final del array.

```
$a[] = "hola"; // $a[2] == "hola"
$a[] = "mundo"; // $a[3] == "mundo"
```

Los arrays se pueden ordenar usando las funciones asort(), arsort(), ksort(), rsort(), sort(), uasort(), usort(), y uksort() dependiendo del tipo de ordenación que se desee.

Se puede contar el número de elementos de un array usando la función count().

Se puede recorrer un array usando las funciones next() y prev(). Otra forma habitual de recorrer un array es usando la función each().

Arrays Multidimensionales

Los arrays multidimensionales son bastante simples actualmente. Para cada dimensión del array, se puede añadir otro valor [clave] al final:

```
$a[1]      = $f;           # ejemplos de una sola dimensión
$a["foo"]  = $f;

$a[1][0]     = $f;         # bidimensional
$a["foo"][2] = $f;         # (se pueden mezclar índices numéricos y asociativos)
$a[3]["bar"] = $f;         # (se pueden mezclar índices numéricos y asociativos)

$a["foo"][4]["bar"][0] = $f;   # tetradimensional!
```

En PHP3 no es posible referirse a arrays multidimensionales directamente dentro de cadenas. Por ejemplo, lo siguiente no tendrá el resultado deseado:

```
$a[3]['bar'] = 'Bob';
echo "Esto no va a funcionar: $a[3][bar]";
```

En PHP3, lo anterior tendrá la salida `Esto no va a funcionar: Array[bar]`. De todas formas, el operador de concatenación de cadenas se puede usar para solucionar esto:

```
$a[3]['bar'] = 'Bob';
echo "Esto no va a funcionar: " . $a[3][bar];
```

En PHP4, sin embargo, todo el problema se puede circunvenir encerrando la referencia al array (dentro de la cadena) entre llaves:

```
$a[3]['bar'] = 'Bob';
echo "Esto va a funcionar: {$a[3][bar]}";
```

Se pueden "rellenar" arrays multidimensionales de muchas formas, pero la más difícil de comprender es cómo usar el comando array() para arrays asociativos. Estos dos trozos de código rellenarán el array unidimensional de la misma manera:

```
# Ejemplo 1:

$a["color"] = "rojo";
$a["sabor"] = "dulce";
$a["forma"] = "redondeada";
$a["nombre"] = "manzana";
```

```
$a[3] = 4;

# Example 2:
$a = array(
     "color" => "rojo",
     "sabor" => "dulce",
     "forma" => "redondeada",
     "nombre"  => "manzana",
     3       => 4
);
```

La función array() se puede anidar para arrays multidimensionales:

```
<?
$a = array(
     "manzana"  => array(
          "color"  => "rojo",
          "sabor"  => "dulce",
          "forma"  => "redondeada"
     ),
     "naranja"  => array(
          "color"  => "naranja",
          "sabor"  => "ácido",
          "forma"  => "redondeada"
     ),
     "plátano"  => array(
          "color"  => "amarillo",
          "sabor"  => "paste-y",
          "forma"  => "aplatanada"
     )
);

echo $a["manzana"]["sabor"];    # devolverá "dulce"
?>
```

Objetos

Inicialización de Objetos

Para inicializar un objeto, se usa la sentencia new para instanciar el objeto a una variable.

```
class foo {
    function do_foo () {
        echo "Doing foo.";
```

```
    }
}

$bar = new foo;
$bar->do_foo();
```

Type juggling

PHP no requiere (o soporta) la declaración explícita del tipo en la declaración de variables; el tipo de una variable se determina por el contexto en el que se usa esa variable. Esto quiere decir que si se asigna un valor de cadena a la variable *var*, *var* se convierte en una cadena. Si después se asigna un valor entero a la variable *var*, se convierte en una variable entera.

Un ejemplo de conversión de tipo automática en PHP3 es el operador suma '+'. Si cualquiera de los operandos es un doble, entonces todos los operandos se evalúan como dobles, y el resultado será un doble. En caso contrario, los operandos se interpretarán como enteros, y el resultado será también un entero. Nótese que esto NO cambia los tipos de los operandos propiamente dichos; el único cambio está en cómo se evalúan los operandos.

```
$foo = "0";  // $foo es una cadena (ASCII 48)
$foo++;      // $foo es la cadena "1" (ASCII 49)
$foo += 1;   // $foo ahora es un entero (2)
$foo = $foo + 1.3;  // $foo ahora es un doble (3.3)
$foo = 5 + "10 Cerditos Pequeñitos"; // $foo es entero (15)
$foo = 5 + "10 Cerditos";     // $foo es entero (15)
```

Si los últimos dos ejemplos anteriores parecen confusos, vea Conversión de cadenas.

Si se desea obligar a que una variable sea evaluada con un tipo concreto, mire la sección Forzado de tipos. Si se desea cambiar el tipo de una variable, vea la función settype().

Si quisiese probar cualquiera de los ejemplos de esta sección, puede cortar y pegar los ejemplos e insertar la siguiente línea para ver por sí mismo lo que va ocurriendo:

```
echo "\$foo==$foo; el tipo es " . gettype( $foo ) . "<br>\n";
```

Nota: La posibilidad de una conversión automática a array no está definida actualmente.

```
$a = 1;       // $a es un entero
$a[0] = "f";  // $a se convierte en un array, en el que $a[0] vale "f"
```

Aunque el ejemplo anterior puede parecer que claramente debería resultar en que $a se convierta en un array, el primer elemento del cual es 'f', consideremos esto:

```
$a = "1";     // $a es una cadena
$a[0] = "f";  // ¿Qué pasa con los índices de las cadenas? ¿Qué ocurre?
```

Dado que PHP soporta indexación en las cadenas vía offsets usando la misma sintaxis que la indexación de arrays, el ejemplo anterior nos conduce a un problema: ¿debería convertirse $a en un array cuyo primer elemento sea "f", o debería convertirse "f" en el primer carácter de la cadena $a?

Por esta razón, tanto en PHP 3.0.12 como en PHP 4.0b3-RC4, el resultado de esta conversión automática se considera que no está definido. Los parches se están discutiendo, de todas formas.

Forzado de tipos

El forzado de tipos en PHP funciona como en C: el nombre del tipo deseado se escribe entre paréntesis antes de la variable a la que se pretende forzar.

```
$foo = 10;   // $foo es un entero
$bar = (double) $foo;   // $bar es un doble
```

Los forzados de tipo permitidos son:

- (int), (integer) - fuerza a entero (integer)
- (real), (double), (float) - fuerza a doble (double)
- (string) - fuerza a cadena (string)
- (array) - fuerza a array (array)
- (object) - fuerza a objeto (object)

Nótese que las tabulaciones y espacios se permiten dentro de los paréntesis, así que los siguientes ejemplos son funcionalmente equivalentes:

```
$foo = (int) $bar;
$foo = ( int ) $bar;
```

Puede no ser obvio que ocurrirá cuando se fuerce entre ciertos tipos. Por ejemplo, lo siguiente debería ser tenido en cuenta.

Cuando se fuerza el cambio de un escalar o una variable de cadena a un array, la variable se convertirá en el primer elemento del array:

```
$var = 'ciao';
$arr = (array) $var;
echo $arr[0];  // produce la salida 'ciao'
```

Cuando se fuerza el tipo de una variable escalar o de una cadena a un objeto, la variable se convertirá en un atributo del objeto; el nombre del atributo será 'scalar':

```
$var = 'ciao';
$obj = (object) $var;
echo $obj->scalar;  // produce la salida 'ciao'
```

Capítulo 8. Variables

Conceptos Básicos

En PHP las variables se representan como un signo de dólar seguido por el nombre de la variable. El nombre de la variable es sensible a minúsculas y mayúsculas.

```
$var = "Bob";
$Var = "Joe";
echo "$var, $Var"; // produce la salida "Bob, Joe"
```

En PHP3, las variables siempre se asignan por valor. Esto significa que cuando se asigna una expresión a una variable, el valor íntegro de la expresión original se copia en la variable de destino. Esto quiere decir que, por ejemplo, después e asignar el valor de una variable a otra, los cambios que se efectúen a una de esas variables no afectará a la otra. Para más información sobre este tipo de asignación, vea Expresiones.

PHP4 ofrece otra forma de asignar valores a las variables: *asignar por referencia*. Esto significa que la nueva variable simplemente referencia (en otras palabras, "se convierte en un alias de" o "apunta a") la variable original. Los cambios a la nueva variable afectan a la original, y viceversa. Esto también significa que no se produce una copia de valores; por tanto, la asignación ocurre más rápidamente. De cualquier forma, cualquier incremento de velocidad se notará sólo en los bucles críticos cuando se asignen grandes arrays u objetos.

Para asignar por referencia, simplemente se antepone un ampersand (&) al comienzo de la variable cuyo valor se está asignando (la variable fuente). Por ejemplo, el siguiente trozo de código produce la salida ’Mi nombre es Bob’ dos veces:

```
<?php
$foo = ’Bob’;              // Asigna el valor ’Bob’ a $foo
$bar = &$foo;              // Referencia $foo vía $bar.
$bar = "Mi nombre es $bar";  // Modifica $bar...
echo $foo;                 // $foo también se modifica.
echo $bar;
?>
```

Algo importante a tener en cuenta es que sólo las variables con nombre pueden ser asignadas por referencia.

```
<?php
$foo = 25;
$bar = &$foo;      // Esta es una asignación válida.
$bar = &(24 * 7);  // Inválida; referencia una expresión sin nombre.

function test() {
   return 25;
}

$bar = &test();    // Inválida.
?>
```

Variables predefinidas

PHP proporciona una gran cantidad de variables predefinidas a cualquier script que se ejecute. De todas formas, muchas de esas variables no pueden estar completamente documentadas ya que dependen de sobre qué servidor se esté ejecutando, la versión y configuración de dicho servidor, y otros factores. Algunas de estas variables no estarán disponibles cuando se ejecute PHP desde la línea de comandos.

A pesar de estos factores, aquí tenemos una lista de variables predefinidas disponibles en una instalación por defecto de PHP 3 corriendo como modulo de un Apache (http://www.apache.org/) 1.3.6 con su configuración también por defecto.

Para una lista de variables predefinidas (y muchas más información útil), por favor, vea (y use) phpinfo().

> **Nota:** Esta lista no es exhaustiva ni pretende serlo. Simplemente es una guía de qué tipo de variables predefinidas se puede esperar tener disponibles en un script.

Variables de Apache

Estas variables son creadas por el servidor web Apache (http://www.apache.org/). Si se está utilizando otro servidor web, no hay garantía de que proporcione las mismas variables; pueden faltar algunas, o proporcionar otras no listadas aquí. Dicho esto, también están presentes las variables de la especificación CGI 1.1 (http://hoohoo.ncsa.uiuc.edu/cgi/env.html), por lo que también se deben tener en cuenta.

Tenga en cuenta que unas pocas, como mucho, de estas variables van a estar disponibles (o simplemente tener sentido) si se ejecuta PHP desde la línea de comandos.

GATEWAY_INTERFACE

: Qué revisión de la especificación CGI está usando el servidor; por ejemplo 'CGI/1.1'.

SERVER_NAME

: El nombre del equipo servidor en el que se está ejecutando el script. Si el script se está ejecutando en un servidor virtual, este será el valor definido para dicho servidor virtual.

SERVER_SOFTWARE

: Una cadena de identificación del servidor, que aparece en las cabeceras al responderse a las peticiones.

SERVER_PROTOCOL

: Nombre y revisión del protocolo a través del que se solicitó la página; p.ej. 'HTTP/1.0';

REQUEST_METHOD

: Qué método de petición se usó para acceder a la página; p.ej. 'GET', 'HEAD', 'POST', 'PUT'.

QUERY_STRING

La cadena de la petición, si la hubo, mediante la que se accedió a la página.

DOCUMENT_ROOT

El directorio raíz del documento bajo el que se ejecuta el script, tal y como está definido en el fichero de configuración del servidor.

HTTP_ACCEPT

Los contenidos de la cabecera `Accept:` de la petición actual, si hay alguna.

HTTP_ACCEPT_CHARSET

Los contenidos de la cabecera `Accept-Charset:` de la petición actual, si hay alguna. Por ejemplo: 'iso-8859-1,*,utf-8'.

HTTP_ENCODING

Los contenidos de la cabecera `Accept-Encoding:` de la petición actual, si la hay. Por ejemplo: 'gzip'.

HTTP_ACCEPT_LANGUAGE

Los contenidos de la cabecera `Accept-Language:` de la petición actual, si hay alguna. Por ejemplo: 'en'.

HTTP_CONNECTION

Los contenidos de la cabecera `Connection:` de la petición actual, si hay alguna. Por ejemplo: 'Keep-Alive'.

HTTP_HOST

Los contenidos de la cabecera `Host:` de la petición actual, si hay alguna.

HTTP_REFERER

La dirección de la página (si la hay) desde la que el navegador saltó a la página actual. Esto lo establece el navegador del usuario; no todos los navegadores lo hacen.

HTTP_USER_AGENT

Los contenidos de la cabecera `User_Agent:` de la petición actual, si hay alguna. Indica el navegador que se está utilizando para ver la página actual; p.ej. `Mozilla/4.5 [en] (X11; U; Linux 2.2.9 i586)`. Entre otras cosas, se puede usar este valor con get_browser() para adaptar la funcionalidad de la página a las posibilidades del navegador del usuario.

REMOTE_ADDR

La dirección IP desde la que el usuario está viendo la página actual.

REMOTE_PORT

El puerto que se está utilizando en la máquina del usuario para comunicarse con el servidor web.

SCRIPT_FILENAME

La vía de acceso absoluta del script que se está ejecutando.

SERVER_ADMIN

El valor que se haya dado a la directiva SERVER_ADMIN (en Apache) en el fichero de configuración del servidor web. Si el script se está ejecutando en un servidor virtual, será el valor definido para dicho servidor virtual.

SERVER_PORT

El puerto del equipo servidor que está usando el servidor web para la comunicación. Para configuraciones por defecto, será ’80’; al usar SSL, por ejemplo, cambiará al puerto que se haya definido como seguro para HTTP.

SERVER_SIGNATURE

Una cadena que contiene la versión del servidor y el nombre del servidor virtual que es añadida a las páginas generadas por el servidor, si está característica está activa.

PATH_TRANSLATED

Vía de acceso basada en el sistema de ficheros- (no el directorio raíz del documento-) del script en cuestión, después de que el servidor haya hecho la conversión virtual-a-real.

SCRIPT_NAME

Contiene la vía de acceso del script actual. Es útil para páginas que necesitan apuntar a sí mismas.

REQUEST_URI

La URI que se dió para acceder a esta página; por ejemplo, ’/index.html’.

Variables de entorno

Estas variables se importan en el espacio de nombres global de PHP desde el entorno en el que se esté ejecutando el intérprete PHP. Muchas son proporcionadas por el intérprete de comandos en el que se está ejecutando PHP, y dado que a sistemas diferentes les gusta ejecutar diferentes tipos de intérpretes de comandos, es imposible hacer una lista definitiva. Por favor, mire la documentación de su intérprete de comandos para ver una lista de las variables de entorno definidas.

Otras variables de entorno son las de CGI, que están ahí sin importar si PHP se está ejecutando como un módulo del servidor o como un intérprete CGI.

Variables de PHP

Estas variables son creadas por el propio PHP.

argv

Array de argumentos pasados al script. Cuando el script se ejecuta desde la línea de comandos, esto da un acceso, al estilo de C, a los parámetros pasados en línea de comandos. Cuando se le llama mediante el método GET, contendrá la cadena de la petición.

argc

Contiene el número de parámetros de la línea de comandos pasados al script (si se ejecuta desde la línea de comandos).

PHP_SELF

El nombre del fichero que contiene el script que se esta ejecutando, relativo al directorio raíz de los documentos. Si PHP se está ejecutando como intérprete de línea de comandos, esta variable no está disponible.

HTTP_COOKIE_VARS

Un array asociativo de variables pasadas al script actual mediante cookies HTTP. Sólo está disponible si el seguimiento de variables ha sido activado mediante la directiva de configuración track_vars o la directiva `<?php_track_vars?>`.

HTTP_GET_VARS

Un array asociativo de variables pasadas al script actual mediante el método HTTP GET. Sólo está disponible si --variable tracking-- ha sido activado mediante la directiva de configuración track_vars o la directiva `<?php_track_vars?>`.

HTTP_POST_VARS

Un array asociativo de variables pasadas al script actual mediante el método HTTP POST. Sólo está disponible si --variable tracking-- ha sido activado mediante la directiva de configuración track_vars o la directiva `<?php_track_vars?>`.

Ambito de las variables

El ámbito de una variable es el contexto dentro del que la variable está definida. La mayor parte de las variables PHP sólo tienen un ámbito simple. Este ámbito simple también abarca los ficheros incluidos y los requeridos. Por ejemplo:

```
$a = 1;
include "b.inc";
```

Aquí, la variable $a dentro del script incluido b.inc. De todas formas, dentro de las funciones definidas por el usuario aparece un ámbito local a la función. Cualquier variables que se use dentro de una función está, por defecto, limitada al ámbito local de la función. Por ejemplo:

```
$a = 1; /* ámbito global */
```

```
Function Test () {
    echo $a; /* referencia a una variable de ámbito local */
}

Test ();
```

Este script no producirá salida, ya que la orden echo utiliza una versión local de la variable $a, a la que no se ha asignado ningún valor en su ámbito. Puede que usted note que hay una pequeña diferencia con el lenguaje C, en el que las variables globales están disponibles automáticamente dentro de la función a menos que sean expresamente sobreescritas por una definición local. Esto puede causar algunos problemas, ya que la gente puede cambiar variables globales inadvertidamente. En PHP, las variables globales deben ser declaradas globales dentro de la función si van a ser utilizadas dentro de dicha función. Veamos un ejemplo:

```
$a = 1;
$b = 2;

Function Sum () {
    global $a, $b;

    $b = $a + $b;
}

Sum ();
echo $b;
```

El script anterior producirá la salida "3". Al declarar $a y $b globales dentro de la función, todas las referencias a tales variables se referirán a la versión global. No hay límite al número de variables globales que se pueden manipular dentro de una función.

Un segundo método para acceder a las variables desde un ámbito global es usando el array $GLOBALS propio de PHP3. El ejemplo anterior se puede reescribir así:

```
$a = 1;
$b = 2;

Function Sum () {
    $GLOBALS["b"] = $GLOBALS["a"] + $GLOBALS["b"];
}

Sum ();
echo $b;
```

El array $GLOBALS es un array asociativo con el nombre de la variable global como clave y los contenidos de dicha variable como el valor del elemento del array.

Otra característica importante del ámbito de las variables es la variable *static*. Una variable estática existe sólo en el ámbito local de la función, pero no pierde su valor cuando la ejecución del programa abandona este ámbito. Consideremos el siguiente ejemplo:

```
Function Test () {
    $a = 0;
    echo $a;
    $a++;
}
```

Esta función tiene poca utilidad ya que cada vez que es llamada asigna a $a el valor 0 y representa un "0". La sentencia $a++, que incrementa la variable, no sirve para nada, ya que en cuanto la función termina la variable $a desaparece. Para hacer una función útil para contar, que no pierda la pista del valor actual del conteo, la variable $a debe declararse como estática:

```
Function Test () {
    static $a = 0;
    echo $a;
    $a++;
}
```

Ahora, cada vez que se llame a la función Test(), se representará el valor de $a y se incrementará.

Las variables estáticas también proporcionan una forma de manejar funciones recursivas. Una función recursiva es la que se llama a sí misma. Se debe tener cuidado al escribir una función recursiva, ya que puede ocurrir que se llame a sí misma indefinidamente. Hay que asegurarse de implementar una forma adecuada de terminar la recursión. La siguiente función cuenta recursivamente hasta 10, usando la variable estática $count para saber cuándo parar:

```
Function Test () {
    static $count = 0;

    $count++;
    echo $count;
    if ($count < 10) {
        Test ();
    }
    $count--;
}
```

Variables variables

A veces es conveniente tener nombres de variables variables. Dicho de otro modo, son nombres de variables que se pueden establecer y usar dinámicamente. Una variable normal se establece con una sentencia como:

```
$a = "hello";
```

Una variable variable toma el valor de una variable y lo trata como el nombre de una variable. En el ejemplo anterior, *hello*, se puede usar como el nombre de una variable utilizando dos signos de dólar. p.ej.

```
$$a = "world";
```

En este momento se han definido y almacenado dos variables en el árbol de símbolos de PHP: $a, que contiene "hello", y $hello, que contiene "world". Es más, esta sentencia:

```
echo "$a ${$a}";
```

produce el mismo resultado que:

```
echo "$a $hello";
```

p.ej. ambas producen el resultado: *hello world*.

Para usar variables variables con arrays, hay que resolver un problema de ambigüedad. Si se escribe $$a[1] el intérprete necesita saber si nos referimos a utilizar $a[1] como una variable, o si se pretendía utilizar $$a como variable y el índice [1] como índice de dicha variable. La sintaxis para resolver esta ambiguedad es: ${$a[1]} para el primer caso y ${$a}[1] para el segundo.

Variables externas a PHP

Formularios HTML (GET y POST)

Cuando se envía un formulario a un script PHP, las variables de dicho formulario pasan a estar automáticamente disponibles en el script gracias a PHP. Por ejemplo, consideremos el siguiente formulario:

Ejemplo 8-1. Variables de formulario simples

```
<form action="foo.php3" method="post">
    Name: <input type="text" name="name"><br>
    <input type="submit">
</form>
```

Cuando es enviado, PHP creará la variable `$name`, que contendrá lo que sea que se introdujo en el campo *Name:* del formulario.

PHP también maneja arrays en el contexto de variables de formularios, pero sólo en una dimensión. Se puede, por ejemplo, agrupar juntas variables relacionadas, o usar esta característica para recuperar valores de un campo select input múltiple:

Ejemplo 8-2. Variables de formulario más complejas

```
<form action="array.php" method="post">
    Name: <input type="text" name="personal[name]"><br>
    Email: <input type="text" name="personal[email]"><br>
    Beer: <br>
    <select multiple name="beer[]">
        <option value="warthog">Warthog
        <option value="guinness">Guinness
        <option value="stuttgarter">Stuttgarter Schwabenbräu
        </select>
    <input type="submit">
</form>
```

Si la posibilidad de PHP de track_vars está activada, ya sea mediante la opción de configuración track_vars o mediante la directiva `<?php_track_vars?>`, las variables enviadas con los métodos POST o GET también se encontrarán en los arrays asociativos globales $HTTP_POST_VARS y $HTTP_GET_VARS.

IMAGE SUBMIT variable names

Cuando se envía un formulario, es posible usar una imagen en vez del botón submit estándar con una etiqueta como:

```
<input type=image src="image.gif" name="sub">
```

Cuando el usuario hace click en cualquier parte de la imagen, el formulario que la acompaña se transmitirá al servidor con dos variables adicionales, sub_x y sub_y. Estas contienen las coordenadas del click del usuario dentro de la imagen. Los más experimentados puede notar que los nombres de variable enviados por el navegador contienen un guión en vez de un subrayado (guión bajo), pero PHP convierte el guión en subrayado automáticamente.

Cookies HTTP

PHP soporta cookies de HTTP de forma transparente tal y como están definidas en en las Netscape’s Spec (http://www.netscape.com/newsref/std/cookie_spec.html). Las cookies son un mecanismo para almacenar datos en el navegador y así rastrear o identificar a usuarios que vuelven. Se pueden crear cookies usando la función **SetCookie()**. Las cookies son parte de la cabecera HTTP, así que se debe llamar a la función SetCookie antes de que se envíe cualquier salida al navegador. Es la misma restricción que para la función header(). Cualquier cookie que se reciba procedente del cliente será convertida automáticamente en una variable de PHP como con los datos en los métodos GET y POST.

Si se quieren asignar múltiples valores a una sola cookie, basta con añadir *[]* al nombre de la. Por ejemplo:

```
SetCookie ("MyCookie[]", "Testing", time()+3600);
```

Nótese que una cookie reemplazará a una cookie anterior que tuviese el mismo nombre en el navegador a menos que el camino (path) o el dominio fuesen diferentes. Así, para una aplicación de carro de la compra se podría querer mantener un contador e ir pasándolo. P.ej.

Ejemplo 8-3. SetCookie Example

```
$Count++;
SetCookie ("Count", $Count, time()+3600);
SetCookie ("Cart[$Count]", $item, time()+3600);
```

Variables de entorno

PHP hace accesibles las variables de entorno automáticamente tratándolas como variables normales.

```
echo $HOME;  /* Shows the HOME environment variable, if set. */
```

Dado que la información que llega vía mecanismos GET, POST y Cookie crean automáticamente variables de PHP, algunas veces es mejor leer variables del entorno explícitamente para asegurarse de que se está trabajando con la versión correcta. La función getenv() se puede usar para ello. También se puede asignar un valor a una variable de entorno con la función putenv().

Puntos en los nombres de variables de entrada

Típicamente, PHP no altera los nombres de las variables cuando se pasan a un script. De todas formas, hay que notar que el punto no es un carácter válido en el nombre de una variable PHP. Por esta razón, mire esto:

```
$varname.ext;  /* nombre de variable no válido */
```

Lo que el intérprete ve es el nombre de una variable $varname, seguido por el operador de concatenación, y seguido por la prueba (es decir, una cadena sin entrecomillar que no coincide con ninguna palabra clave o reservada conocida) 'ext'. Obviamente, no se pretendía que fuese este el resultado.

Por esta razón, es importante hacer notar que PHP reemplazará automáticamente cualquier punto en los nombres de variables de entrada por guiones bajos (subrayados).

Determinando los tipos de variables

Dado que PHP determina los tipos de las variables y los convierte (generalmente) según necesita, no siempre resulta obvio de qué tipo es una variable dada en un momento concreto. PHP incluye varias funciones que descubren de qué tipo es una variable. Son gettype(), is_long(), is_double(), is_string(), is_array(), y is_object().

Capítulo 9. Constantes

Una constante es un identificador para expresar un valor simple. Como el nombre sugiere, este valor no puede variar durante la ejecución del script. (Las constantes especiales `__FILE__` y `__LINE__` son una excepción a esto, ya que actualmente no lo soin). Una constante es sensible a mayúsculas por defecto. Por convención, los identificadores de constantes suelen declararse en mayúsculas

El nombre de una constante sigue las mismas reglas que cualquier etiqueta en PHP. Un nombre de constante válido empieza con una letra o un caracter de subrayado, seguido por cualquier número de letras, números, o subrayados. Se podrían expresar mediante la siguiente expresión regular: `[a-zA-Z_\x7f-\xff][a-zA-Z0-9_\x7f-\xff]*`

> **Nota:** Para nuestros propósitos , entenderemos como letra los carácteres a-z, A-Z, y los ASCII del 127 hasta el 255 (0x7f-0xff).

El alcanze de una constante es global, Es decir, es posible acceder a ellas sin preocuparse por el ámbito de alcance.

Sintaxis

Se puede definir una constante usando la función define(). Una vez definida, no puede ser modificada ni eliminada .

Solo se puede definir como constantes valores escalares (boolean, integer, float y string).

Para obtener el valor de una constante solo es necesario especificar su nombre. A diferencia de las variables, *no* se tiene que especificar el prefijo `$`. Tambien se puede utilizar la función constant(), para obtener el valor de una constante, en el caso de que queramos expresarla de forma dinámica Usa la función get_defined_constants() parar obtener una lista de todas las constantes definidas.

> **Nota:** Las contantes y las variables (globales) se encuentran en un espacio de nombres distinto. Esto implica que por ejemplo `TRUE` y `$TRUE` son diferentes.

Si usas una constante todavia no definida, PHP asume que estás refiriéndote al nombre de la constante en si. Se lanzará un aviso si esto sucede. Usa la función defined() para comprobar la existencia de dicha constante.

Estas son las diferencias entre constantes y variables:

- Las constantes no son precedidas por un símbolo de dolar (`$`)
- Las contantes solo pueden ser definidas usando la **función()** define , nunca por simple asignación
- Las constantes pueden ser definidas y accedidas sin tener en cuenta las reglas de alcanze del ámbito.
- Las constantes no pueden ser redefinidas o eliminadas despues de establecerse; y
- Las constantes solo puede albergar valores escalares

Ejemplo 9-1. Definiendo constantes

```
<?php
define("CONSTANT", "Hello world.");
echo CONSTANT; // outputs "Hello world."
echo Constant; // outputs "Constant" and issues a notice.
?>
```

Constantes predefinidas

PHP ofrece un largo número de constantes predefinidas a cualquier script en ejecución. Muchas de estas constantes, sin embargo, son creadas por diferentes extensiones, y solo estarán presentes si dichas extensiones están disponibles, bien por carga dinámica o porque has sido compiladas.

Se puede encontrar una lista de constantes predefinidas en la seccion Constantes predefinidas.

Capítulo 10. Expresiones

Las expresiones son la piedra angular de PHP. En PHP, casi cualquier cosa que escribes es una expresión. La forma más simple y ajustada de definir una expresión es "cualquier cosa que tiene un valor".

Las formas más básicas de expresiones son las constantes y las variables. Cuando escribes "$a = 5", estás asignando ’5’ a $a. ’5’, obviamente, tiene el valor 5 o, en otras palabras ’5’ es una expresión con el valor 5 (en este caso, ’5’ es una constante entera).

Después de esta asignación, esperarás que el valor de $a sea 5 también, de manera que si escribes $b = $a, esperas que se comporte igual que si escribieses $b = 5. En otras palabras, $a es una expresión también con el valor 5. Si todo va bien, eso es exactamente lo que pasará.

Las funciones son un ejemplo algo más complejo de expresiones. Por ejemplo, considera la siguiente función:

```
function foo () {
    return 5;
}
```

Suponiendo que estés familiarizado con el concepto de funciones (si no lo estás échale un vistazo al capítulo sobre funciones), asumirás que teclear `$c = foo()` es esencialmente lo mismo que escribir `$c = 5`, y has acertado. Las funciones son expresiones que valen el valor que retornan. Como foo() devuelve 5, el valor de la expresión ’foo()’ es 5. Normalmente las funciones no devuelven un valor fijo, sino que suele ser calculado.

Desde luego, los valores en PHP no se limitan a enteros, y lo más normal es que no lo sean. PHP soporta tres tipos escalares: enteros, punto flotante y cadenas (los tipos escalares son aquellos cuyos valores no pueden ’dividirse’ en partes menores, no como los arrays, por ejemplo). PHP también soporta dos tipos compuestos (no escalares): arrays y objetos. Se puede asignar cada uno de estos tipos de valor a variables o bien retornarse de funciones, sin ningún tipo de limitación.

Hasta aquí, los usuarios de PHP/FI 2 no deberían haber notado ningún cambio. Sin embargo, PHP lleva las expresiones mucho más allá, al igual que otros lenguajes. PHP es un lenguaje orientado a expresiones, en el sentido de que casi todo es una expresión. Considera el ejemplo anterior ’$a = 5’. Es sencillo ver que hay dos valores involucrados, el valor de la constante entera ’5’, y el valor de $a que está siendo actualizado también a 5. Pero la verdad es que hay un valor adicional implicado aquí, y es el valor de la propia asignación. La asignación misma se evalúa al valor asignado, en este caso 5. En la práctica, quiere decir que ’$a = 5’, independientemente de lo que hace, es una expresión con el valor 5. De esta manera, escribir algo como ’$b = ($a = 5)’ es como escribir ’$a = 5; $b = 5;’ (un punto y coma marca el final de una instrucción). Como las asignaciones se evalúan de derecha a izquierda, puedes escribir también ’$b = $a = 5’.

Otro buen ejemplo de orientación a expresiones es el pre y post incremento y decremento. Los usuarios de PHP/FI 2 y los de otros muchos lenguajes les sonará la notación variable++ y variable--. Esto son las operaciones de incremento y decremento. En PHP/FI 2, la instrucción ’$a++’ no tiene valor (no es una expresión), y no puedes asignarla o usarla de ningún otro modo. PHP mejora las características del incremento/decremento haciéndolos también expresiones, como en C. En PHP, como en C, hay dos tipos de incremento - pre-incremento y post-incremento. Ambos, en esencia, incrementan la variable y el efecto en la variable es idéntico. La diferencia radica en el valor de la propia expresion incremento. El preincremento , escrito ’++$variable’, se evalúa al valor incrementado (PHP incrementa la variable antes de leer su valor, de ahí el nombre ’preincremento’). El postincremento, escrito ’$variable++’, se evalúa al

valor original de $variable antes de realizar el incremento (PHP incrementa la variable después de leer su valor, de ahí el nombre 'postincremento').

Un tipo muy corriente de expresiones son las expresiones de comparación. Estas expresiones se evalúan a 0 o 1, significando FALSO (`FALSE`) o CIERTO (`TRUE`), respectivamente. PHP soporta > (mayor que), >= (mayor o igual que), == (igual que), != (distinto), < (menor que) y <= (menor o igual que). Estas expresiones se usan frecuentemente dentro de la ejecución condicional como la instrucción `if`.

El último tipo de expresiones que trataremos, es la combinación operador-asignación. Ya sabes que si quieres incrementar $a en 1, basta con escribir '$a++' o ++$a'. Pero qué pasa si quieres añadir más de 1, por ejemplo 3? Podrías escribir '$a++' múltiples veces, pero no es una forma de hacerlo ni eficiente ni cómoda. Una práctica mucho más corriente es escribir '$a = $a + 3'. '$a + 3' se evalúa al valor de $a más 3, y se asigna de nuevo a $a, lo que resulta en incrementar $a en 3. En PHP, como en otros lenguajes como C, puedes escribir esto de una forma más concisa, que con el tiempo será más clara y también fácil de entender. Añadir 3 al valor actual de $a se puede escribir como '$a += 3'. Esto quiere decir exactamente "toma el valor de $a, súmale 3, y asígnalo otra vez a $a". Además de ser más corto y claro, también resulta en una ejecución más rápida. El valor de '$a += 3', como el valor de una asignación normal y corriente, es el valor asignado. Ten en cuenta que NO es 3, sino el valor combinado de $a más 3 (ése es el valor asignado a $a). Cualquier operación binaria puede ser usada en forma de operador-asignación, por ejemplo '$a -= 5' (restar 5 del valor de $a), '$b *= 7' (multiplicar el valor de $b por 5), etc.

Hay otra expresión que puede parecer extraña si no la has visto en otros lenguaes, el operador condicional ternario:

```
$first ? $second : $third
```

Si el valor de la primera subexpresión es verdadero (distinto de cero), entonces se evalúa la segunda subexpresión, si no, se evalúa la tercera y ése es el valor.

El siguiente ejemplo te ayudará a comprender un poco mejor el pre y post incremento y las expresiones en general:

```
function double($i) {
    return $i*2;
}
$b = $a = 5;        /* asignar el valor cinco a las variables $a y $b */
$c = $a++;          /* postincremento, asignar el valor original de $a (5) a $c */
$e = $d = ++$b;     /* preincremento, asignar el valor incrementado de $b (6) a
                       $d y a $e */

/* en este punto, tanto $d como $e son iguales a 6 */

$f = double($d++);  /* asignar el doble del valor de $d antes
                       del incremento, 2*6 = 12 a $f */
$g = double(++$e);  /* asignar el doble del valor de $e después
                       del incremento, 2*7 = 14 a $g */
$h = $g += 10;      /* primero, $g es incrementado en 10 y termina valiendo 24.
                       después el valor de la asignación (24) se asigna a $h,
                       y $h también acaba valiendo 24. */
```

Al principio del capítulo hemos dicho que describiríamos los distintos tipos de instrucciones y, como prometimos, las expresiones pueden ser instrucciones. Sin embargo, no todas las expresiones son instrucciones. En este caso, una instrucción tiene la forma 'expr' ';', es decir, una expresión seguida de un punto y coma. En '$b=$a=5;', $a=5 es una expresión válida, pero no es una instrucción en sí misma. Por otro lado '$b=$a=5:' sí es una instrucción válida.

Una última cosa que vale la pena mencionar, es el valor booleano de las expresiones. En muchas ocasiones, principalmente en condicionales y bucles, no estás interesado en el valor exacto de la expresión, sino únicamente si es CIERTA (`TRUE`) o FALSA (`FALSE`) (PHP no tiene un tipo booleano específico). El valor de verdad de las expresiones en PHP se calcula de forma similar a perl. Cualquier valor numérico distinto de cero es CIERTO (`TRUE`), cero es FALSO (`FALSE`). Fíjate en que los valores negativos son distinto de cero y considerados CIERTO (`TRUE`)! La cadena vacía y la cadena "0" son FALSO (`FALSE`); todas las demás cadenas son `TRUE`. Con los tipos no escalares (arrays y objetos) - si el valor no contiene elementos se considera FALSO (`FALSE`), en caso contrario se considera CIERTO (`TRUE`).

PHP te brinda una completa y potente implementación de expresiones, y documentarla enteramente está más allá del objetivo de ete manual. Los ejemplos anteriores, deberían darte una buena idea de qué son las expresiones y cómo construir expresiones útiles. A lo largo del resto del manual, escribiremos *`expr`* para indicar una expresión PHP válida.

Capítulo 11. Operadores

Operadores Aritméticos

¿Recuerdas la aritmética básica del colegio? Pues estos operadores funcionan exactamente igual.

Tabla 11-1. Operadores Aritméticos

ejemplo	nombre	resultado
$a + $b	Adición	Suma de $a y $b.
$a - $b	Substracción	Diferencia entre $a y $b.
$a * $b	Multiplicación	Producto de $a and $b.
$a / $b	División	Cociente de $a entre $b.
$a % $b	Módulo	Resto de $a dividido entre $b.

Operadores de Asignación

El operador básico de asignación es "=". A primera vista podrías pensar que es el operador de comparación "igual que". Pero no. Realmente significa que el operando de la izquierda toma el valor de la expresión a la derecha, (esto es, "toma el valor de").

El valor de una expresión de asignación es el propio valor asignado. Esto es, el valor de "$a = 3" es 3. Esto permite hacer cosas curiosas como

```
$a = ($b = 4) + 5; // ahora $a es igual a 9, y $b vale 4.
```

Además del operador básico de asignación, existen los "operadores combinados" para todas las operaciones aritméticas y de cadenas que sean binarias. Este operador combinado te permite, de una sola vez, usar una variable en una expresión y luego establecer el valor de esa variable al resultado de la expresión. Por ejemplo:

```
$a = 3;
$a += 5; // establece $a a 8, como si hubiésemos escrito: $a = $a + 5;
$b = "Hola ";
$b .= "Ahí!"; // establece $b a "Hola Ahí!", igual que si hiciésemos $b = $b . "Ahí!";
```

Fíjate en que la asignación realiza una nueva copia de la variable original (asignación por valor), por lo que cambios a la variable original no afectan a la copia. Esto puede tener interés si necesitas copiar algo como un array con muchos elementos dentro de un bucle que se repita muchas veces (cada vez se realizará una nueva copia del array). PHP4 soporta asignación por referencia, usando la sintaxis `$var = &$othervar;`, pero esto no es posible en PHP3. ’Asignación por referencia’ quiere decir que ambas variables acabarán apuntando al mismo dato y que nada es realmente copiado.

Operadores Bit a bit

Los operadores bit a bit te permiten activar o desactivar bits individuales de un entero.

Tabla 11-2. Operadores Bit a bit

ejemplo	nombre	resultado
$a & $b	Y	Se activan los bits que están activos tanto en $a como $b.
$a | $b	O	Se activan los bits que están activos en $a o que lo están en $b.
$a ^ $b	Xor ("o exclusiva")	Se activan los bits que están activos en $a o en $b pero no en ambos a la vez.
~ $a	No	Se activan los bits que no están activos en $a.
$a << $b	Desplazamiento a la izquierda	Desplaza los bits de $a, $b posiciones hacia la izquierda (por aritmética binaria, cada posición desplazada equivale a multiplicar por dos el valor de $a)
$a >> $b	Desplazamiento a la derecha	Desplaza los bits de $a, $b posiciones hacia la derecha (por aritmética binaria, cada posición desplazada equivale a dividir entre dos el valor de $a)

Operadores de Comparación

Los operadores de comparación, como su nombre indica, permiten comparar dos valores.

Tabla 11-3. Operadores de Comparación

ejemplo	nombre	resultado
$a == $b	Igualdad	Cierto si $a es igual a $b.
$a === $b	Identidad	Cierto si $a es igual a $b y si son del mismo tipo (sólo PHP4)
$a != $b	Desigualdad	Cierto si $a no es igual a $b.
$a < $b	Menor que	Cierto si $a es estrictamente menor que $b.
$a > $b	Mayor que	Cierto si $a es estrictamente mayor que $b.
$a <= $b	Menor o igual que	Cierto si $a es menor o igual que $b.

ejemplo	nombre	resultado
$a >= $b	Mayor o igual que	Cierto si $a es mayor o igual que $b.

Otro operador condicional es el operador "?:" (o ternario), que funciona como en C y otros muchos lenguajes.

```
(expr1) ? (expr2) : (expr3);
```

La expresión toma el valor *expr2* si *expr1* se evalúa a cierto, y *expr3* si *expr1* se evalúa a falso.

Operador de ejecución

PHP soporta un operador de ejecución: el apóstrofe invertido (“). ¡Fíjate que no son apostrofes normales! PHP intentará ejecutar la instrucción contenida dentro de los apóstrofes invertidos como si fuera un comando del shell; y su salida devuelta como el valor de esta expresión (i.e., no tiene por qué ser simplemente volcada como salida; puede asignarse a una variable).

```
$output = ‘ls -al‘;
echo "<pre>$output</pre>";
```

Ver también system(), passthru(), exec(), popen(), y escapeshellcmd().

Operadores de Incremento/decremento

PHP soporta los operadores de predecremento y post incremento al estilo de C.

Tabla 11-4. Operadores de Incremento/decremento

ejemplo	nombre	efecto
++$a	Preincremento	Incrementa $a en uno y después devuelve $a.
$a++	Postincremento	Devuelve $a y después incrementa $a en uno.
--$a	Predecremento	Decrementa $a en uno y después devuelve $a.
$a--	Postdecremento	Devuelve $a y después decrementa $a en uno.

He aquí un listado de ejemplo:

```
<?php
echo "<h3>Postincremento</h3>";
$a = 5;
echo "Debería ser 5: " . $a++ . "<br>\n";
echo "Debería ser 6: " . $a . "<br>\n";

echo "<h3>Preincremento</h3>";
$a = 5;
echo "Debería ser 6: " . ++$a . "<br>\n";
echo "Debería ser 6: " . $a . "<br>\n";

echo "<h3>Postdecremento</h3>";
$a = 5;
echo "Debería ser 5: " . $a-- . "<br>\n";
echo "Debería ser 4: " . $a . "<br>\n";

echo "<h3>Predecremento</h3>";
$a = 5;
echo "Debería ser 4: " . --$a . "<br>\n";
echo "Debería ser 4: " . $a . "<br>\n";
?>
```

Operadores Lógicos

Tabla 11-5. Operadores Lógicos

ejemplo	nombre	resultado
$a and $b	Y	Cierto si tanto $a como $b son ciertos.
$a or $b	O	Cierto si $a o $b son ciertos.
$a xor $b	O exclusiva	Cierto si $a es cierto o $b es cierto, pero no ambos a la vez.
! $a	Negación	Cierto si $a no es cierto.
$a && $b	Y	Cierto si tanto $a como $b son ciertos.
$a || $b	O	Cierto si $a o $b son ciertos.

La razón de las dos variaciones de "y" y "o" es que operan con distinta precedencia (ver Precedencia de Operadores.)

Precedencia de Operadores

La precedencia de operadores especifica cómo se agrupan las expresiones. Por ejemplo, en la expresión `1`

`+ 5 * 3`, la respuesta es 16 y no 18 porque el operador de multiplicación ("*") tiene una mayor precedencia que el de adición ("+").

La siguiente tabla lista la precedencia de operadores, indicándose primero los de menor precedencia.

Tabla 11-6. Precedencia de Operadores

<table>
<tr><th>Asociatividad</th><th>Operadores</th></tr>
<tr><td>izquierda</td><td>,</td></tr>
<tr><td>izquierda</td><td>or</td></tr>
<tr><td>izquierda</td><td>xor</td></tr>
<tr><td>izquierda</td><td>and</td></tr>
<tr><td>derecha</td><td>print</td></tr>
<tr><td>izquierda</td><td>= += -= *= /= .= %= &= |= ^= ~= <<= >>=</td></tr>
<tr><td>izquierda</td><td>? :</td></tr>
<tr><td>izquierda</td><td>||</td></tr>
<tr><td>izquierda</td><td>&&</td></tr>
<tr><td>izquierda</td><td>|</td></tr>
<tr><td>izquierda</td><td>^</td></tr>
<tr><td>izquierda</td><td>&</td></tr>
<tr><td>no asociativo</td><td>== != ===</td></tr>
<tr><td>no asociativo</td><td>< <= > >=</td></tr>
<tr><td>izquierda</td><td><< >></td></tr>
<tr><td>izquierda</td><td>+ - .</td></tr>
<tr><td>izquierda</td><td>* / %</td></tr>
<tr><td>derecha</td><td>! ~ ++ -- (int) (double) (string) (array) (object) @</td></tr>
<tr><td>derecha</td><td>[</td></tr>
<tr><td>no asociativo</td><td>new</td></tr>
</table>

Operadores de Cadenas

Hay dos operadores de cadenas. El primero es el operador de concatenación ('.'), que devuelve el resultado de concatenar sus operandos izquierdo y derecho. El segundo es el operador de concatenación y asignación ('.='). Consulta Operadores de Asignación para más información.

```
$a = "Hola ";
$b = $a . "Mundo!"; // ahora $b contiene "Hola Mundo!"

$a = "Hola ";
$a .= "Mundo!"; // ahora $a contiene "Hola Mundo!"
```

Capítulo 12. Estructuras de Control

Todo archivo de comandos PHP se compone de una serie de sentencias. Una sentencia puede ser una asignación, una llamada a función, un bucle, una sentencia condicional e incluso una sentencia que no haga nada (una sentencia vacía). Las sentencias normalmente acaban con punto y coma. Además, las sentencias se pueden agrupar en grupos de sentencias encapsulando un grupo de sentencias con llaves. Un grupo de sentencias es también una sentencia. En este capítulo se describen los diferentes tipos de sentencias.

`if`

La construcción `if` es una de las más importantes características de muchos lenguajes, incluido PHP. Permite la ejecución condicional de fragmentos de código. PHP caracteriza una estructura `if` que es similar a la de C:

```
if (expr)
    sentencia
```

Como se describe en la sección sobre expresiones, expr se evalúa a su valor condicional. Si *`expr`* se evalúa como `TRUE`, PHP ejecutará la sentencia, y si se evalúa como `FALSE` - la ignorará.

El siguiente ejemplo mostraría `a es mayor que b` si *`$a`* fuera mayor que *`$b`*:

```
if ($a > $b)
    print "a es mayor que b";
```

A menudo, se desea tener más de una sentencia ejecutada de forma condicional. Por supuesto, no hay necesidad de encerrar cada sentencia con una cláusula `if`. En vez de eso, se pueden agrupar varias sentencias en un grupo de sentencias. Por ejemplo, este código mostraría `a es mayor que b` si *`$a`* fuera mayor que *`$b`*, y entonces asignaría el valor de *`$a`* a *`$b`*:

```
if ($a > $b) {
    print "a es mayor que b";
    $b = $a;
}
```

Las sentencias if se pueden anidar indefinidamente dentro de otras sentencias `if`, lo cual proporciona una flexibilidad completa para ejecuciones condicionales en las diferentes partes de tu programa.

`else`

A menudo queremos ejecutar una sentencia si se cumple una cierta condicion, y una sentencia distinta si la condición no se cumple. Esto es para lo que sirve `else`. `else` extiende una sentencia `if` para ejecutar

una sentencia en caso de que la expresión en la sentencia `if` se evalúe como `FALSE`. Por ejemplo, el siguiente código mostraría `a es mayor que b` si *$a* fuera mayor que *$b*, y `a NO es mayor que b` en cualquier otro caso:

```
if ($a > $b) {
    print "a es mayor que b";
} else {
    print "a NO es mayor que b";
}
```

La sentencia `else` se ejecuta solamente si la expresión `if` se evalúa como `FALSE`, y si hubiera alguna expresión `elseif` - sólo si se evaluaron también a `FALSE` (Ver elseif).

elseif

`elseif`, como su nombre sugiere, es una combinación de `if` y `else`. Como `else`, extiende una sentencia `if` para ejecutar una sentencia diferente en caso de que la expresión `if` original se evalúa como `FALSE`. No obstante, a diferencia de `else`, ejecutará esa expresión alternativa solamente si la expresión condicional `elseif` se evalúa como `TRUE`. Por ejemplo, el siguiente código mostraría `a es mayor que b`, `a es igual a b` o `a es menor que b`:

```
if ($a > $b) {
    print "a es mayor que b";
} elseif ($a == $b) {
    print "a es igual que b";
} else {
    print "a es mayor que b";
}
```

Puede haber varios `elseif`s dentro de la misma sentencia `if`. La primera expresión `elseif` (si hay alguna) que se evalúe como `TRUE` se ejecutaría. En PHP, también se puede escribir 'else if' (con dos palabras) y el comportamiento sería idéntico al de un 'elseif' (una sola palabra). El significado sintáctico es ligeramente distinto (si estas familiarizado con C, es el mismo comportamiento) pero la línea básica es que ambos resultarían tener exactamente el mismo comportamiento.

La sentencia `elseif` se ejecuta sólo si la expresión `if` precedente y cualquier expresión `elseif` precedente se evalúan como `FALSE`, y la expresión `elseif` actual se evalúa como `TRUE`.

Sintaxis Alternativa de Estructuras de Control

PHP ofrece una sintaxis altenativa para alguna de sus estructuras de control; a saber, `if`, `while`, `for`, y `switch`. En cada caso, la forma básica de la sintaxis alternativa es cambiar abrir-llave por dos puntos (:)

y cerrar-llave por `endif;`, `endwhile;`, `endfor;`, or `endswitch;`, respectivamente.

```
<?php if ($a==5): ?>
A es igual a 5
<?php endif; ?>
```

En el ejemplo de arriba, el bloque HTML "A = 5" se anida dentro de una sentencia `if` escrita en la sintaxis alternativa. El bloque HTML se mostraría solamente si $a fuera igual a 5.

La sintaxis alternativa se aplica a `else` y también a `elseif`. La siguiente es una estructura `if` con `elseif` y `else` en el formato alternativo:

```
if ($a == 5):
    print "a es igual a 5";
    print "...";
elseif ($a == 6):
    print "a es igual a 6";
    print "!!!";
else:
    print "a no es ni 5 ni 6";
endif;
```

Mirar también while, for, e if para más ejemplos.

while

Los bucles `while` son los tipos de bucle más simples en PHP. Se comportan como su contrapartida en C. La forma básica de una sentencia `while` es:

```
while (expr) sentencia
```

El significado de una sentencia `while` es simple. Le dice a PHP que ejecute la(s) sentencia(s) anidada(s) repetidamente, mientras la expresión `while` se evalúe como `TRUE`. El valor de la expresión es comprobado cada vez al principio del bucle, así que incluso si este valor cambia durante la ejecución de la(s) sentencia(s) anidada(s), la ejecución no parará hasta el fin de la iteración (cada vez que PHP ejecuta las sentencias en el bucle es una iteración). A veces, si la expresión `while` se evalúa como `FALSE` desde el principio de todo, la(s) sentencia(s) anidada(s) no se ejecutarán ni siquiera una vez.

Como con la sentencia `if`, se pueden agrupar multiples sentencias dentro del mismo bucle `while` encerrando un grupo de sentencias con llaves, o usando la sintaxis alternativa:

```
while (expr): sentencia ... endwhile;
```

Los siguientes ejemplos son idénticos, y ambos imprimen números del 1 al 10:

```
/* ejemplo 1 */

$i = 1;
while ($i <= 10) {
    print $i++;  /* el valor impreso sería
                    $i antes del incremento
                    (post-incremento) */
}

/* ejemplo 2 */

$i = 1;
while ($i <= 10):
    print $i;
    $i++;
endwhile;
```

do..while

Los bucles `do..while` son muy similares a los bucles `while`, excepto que las condiciones se comprueban al final de cada iteración en vez de al principio. La principal diferencia frente a los bucles regulares `while` es que se garantiza la ejecución de la primera iteración de un bucle `do..while` (la condición se comprueba sólo al final de la iteración), mientras que puede no ser necesariamente ejecutada con un bucle `while` regular (la condición se comprueba al principio de cada iteración, si esta se evalúa como `FALSE` desde el principio la ejecución del bucle finalizará inmediatamente).

Hay una sola sintaxis para los bucles `do..while`:

```
$i = 0;
do {
    print $i;
} while ($i>0);
```

El bucle de arriba se ejecutaría exactamente una sola vez, después de la primera iteración, cuando la condición se comprueba, se evalúa como `FALSE` ($i no es más grande que 0) y la ejecución del bucle finaliza.

Los usuarios avanzados de C pueden estar familiarizados con un uso distinto del bucle `do..while`, para permitir parar la ejecución en medio de los bloques de código, encapsulandolos con `do..while`(0), y usando la sentencia `break`. El siguiente fragmento de código demuestra esto:

```
do {
    if ($i < 5) {
        print "i no es lo suficientemente grande";
        break;
    }
    $i *= $factor;
    if ($i < $minimum_limit) {
        break;
    }
    print "i es correcto";
    ...procesa i...
} while(0);
```

No se preocupes si no entiende esto completamente o en absoluto. Se pueden codificar archivos de comandos e incluso archivos de comandos potentes sin usar esta 'propiedad'.

`for`

Los bucles `for` son los bucles más complejos en PHP. Se comportan como su contrapartida en C. La sintaxis de un bucle `for` es:

```
for (expr1; expr2; expr3) sentencia
```

La primera expresión (*`expr1`*) se evalúa (ejecuta) incondicionalmente una vez al principio del bucle.

Al comienzo de cada iteración, se evalúa *`expr2`* . Si se evalúa como `TRUE`, el bucle continúa y las sentencias anidadas se ejecutan. Si se evalúa como `FALSE`, la ejecución del bucle finaliza.

Al final de cada iteración, se evalúa (ejecuta) *`expr3`*.

Cada una de las expresiones puede estar vacía. Que *`expr2`* esté vacía significa que el bucle debería correr indefinidamente (PHP implicitamente lo considera como `TRUE`, al igual que C). Esto puede que no sea tan inútil como se podría pensar, puesto que a menudo se quiere salir de un bucle usando una sentencia `break` condicional en vez de usar la condición de `for`.

Considera los siguientes ejemplos. Todos ellos muestran números del 1 al 10:

```
/* ejemplo 1 */

for ($i = 1; $i <= 10; $i++) {
    print $i;
}

/* ejemplo 2 */

for ($i = 1;;$i++) {
    if ($i > 10) {
```

```
        break;
    }
    print $i;
}

/* ejemplo 3 */

$i = 1;
for (;;) {
    if ($i > 10) {
        break;
    }
    print $i;
    $i++;
}

/* ejemplo 4 */

for ($i = 1; $i <= 10; print $i, $i++) ;
```

Por supuesto, el primer ejemplo parece ser el mas elegante (o quizás el cuarto), pero uno puede descubrir que ser capaz de usar expresiones vacías en bucles `for` resulta útil en muchas ocasiones.

PHP también soporta la "sintaxis de dos puntos" alternativa para bucles `for`.

```
for (expr1; expr2; expr3): sentencia; ...; endfor;
```

Otros lenguajes poseen una sentencia `foreach` para traducir un array o una tabla hash. PHP3 no posee tal construcción; PHP4 sí (ver foreach). En PHP3, se puede combinar while con las funciones list() y each() para conseguir el mismo efecto. Mirar la documentación de estas funciones para ver un ejemplo.

foreach

PHP4 (PHP3 no) incluye una construcción `foreach`, tal como perl y algunos otros lenguajes. Esto simplemente da un modo fácil de iterar sobre arrays. Hay dos sintaxis; la segunda es una extensión menor, pero útil de la primera:

```
foreach(expresion_array as $value) sentencia
foreach(expresion_array as $key => $value) sentencia
```

La primera forma recorre el array dado por `expresion_array`. En cada iteración, el valor del elemento actual se asigna a `$value` y el puntero interno del array se avanza en una unidad (así en el siguiente paso, se estará mirando el elemento siguiente).

La segunda manera hace lo mismo, salvo que la clave del elemento actual será asignada a la variable `$key` en cada iteración.

> **Nota:** Cuando `foreach` comienza su primera ejecución, el puntero interno a la lista (array) se reinicia automáticamente al primer elemento del array. Esto significa que no se necesita llamar a reset() antes de un bucle `foreach`.

> **Nota:** Hay que tener en cuanta que`foreach` con una copia de la lista (array) especificada y no la lista en si, por ello el puntero de la lista no es modificado como en la construcción each.

Puede haber observado que las siguientes son funcionalidades idénticas:

```
reset( $arr );
while( list( , $value ) = each( $arr ) ) {
   echo "Valor: $value<br>\n";
}

foreach( $arr as $value ) {
   echo "Valor: $value<br>\n";
}
```

Las siguientes también son funcionalidades idénticas:

```
reset( $arr );
while( list( $key, $value ) = each( $arr ) ) {
   echo "Key: $key; Valor: $value<br>\n";
}

foreach( $arr as $key => $value ) {
   echo "Key: $key; Valor: $value<br>\n";
}
```

Algunos ejemplos más para demostrar su uso:

```
/* foreach ejemplo 1: sólo valor*/
$a = array(1, 2, 3, 17);

foreach($a as $v) {
   print "Valor actual de \$a: $v.\n";
}
```

```
/* foreach ejemplo 2: valor (con clave impresa para ilustrar) */
$a = array(1, 2, 3, 17);

$i = 0; /* sólo para propósitos demostrativos */

foreach($a as $v) {
   print "\$a[$i] => $k.\n";
}

/* foreach ejemplo 3: clave y valor */
$a = array(
   "uno" => 1,
   "dos" => 2,
   "tres" => 3,
   "diecisiete" => 17
);

foreach($a as $k => $v) {
   print "\$a[$k] => $v.\n";
}
```

break

`break` escapa de la estructuras de control iterante (bucle) actuales `for`, `while`, o `switch`.

`break` accepta un parámetro opcional, el cual determina cuantas estructuras de control hay que escapar.

```
$arr = array ('one', 'two', 'three', 'four', 'stop', 'five');
while (list (, $val) = each ($arr)) {
    if ($val == 'stop') {
        break;    /* You could also write 'break 1;' here. */
    }
    echo "$val<br>\n";
}

/* Using the optional argument. */

$i = 0;
while (++$i) {
    switch ($i) {
    case 5:
        echo "At 5<br>\n";
        break 1;  /* Exit only the switch. */
    case 10:
        echo "At 10; quitting<br>\n";
        break 2;  /* Exit the switch and the while. */
    default:
```

```
        break;
    }
}
```

continue

`continue` se usa dentro de la estructura del bucle para saltar el resto de la iteración actual del bucle y continuar la ejecución al comienzo de la siguiente iteración.

`continue` accepta un parámetro opcional, el cual determina cuantos niveles (bluces) hay que saltar antes de continuar con la ejecución.

```
 while (list($key,$value) = each($arr)) {
     if ($key % 2) { // salta los miembros impares
         continue;
     }
     do_something_odd ($value);
 }
$i = 0;
while ($i++ < 5) {
    echo "Outer<br>\n";
    while (1) {
        echo "  Middle<br>\n";
        while (1) {
            echo "  Inner<br>\n";
            continue 3;
        }
        echo "This never gets output.<br>\n";
    }
    echo "Neither does this.<br>\n";
}
```

switch

La sentencia `switch` es similar a una serie de sentencias IF en la misma expresión. En muchas ocasiones, se quiere comparar la misma variable (o expresión) con nuchos valores diferentes, y ejecutar una parte de código distinta dependiendo de a qué valor es igual. Para ello sirve la sentencia `switch`.

Los siguientes dos ejemplos son dos modos distintos de escribir la misma cosa, uno usa una serie de sentencias `if`, y el otro usa la sentencia `switch`:

```
 if ($i == 0) {
     print "i es igual a 0";
```

```
}
if ($i == 1) {
    print "i es igual a 1";
}
if ($i == 2) {
    print "i es igual a 2";
}

switch ($i) {
    case 0:
        print "i es igual a 0";
        break;
    case 1:
        print "i es igual a 1";
        break;
    case 2:
        print "i es igual a 2";
        break;
}
```

Es importante entender cómo se ejecuta la sentencia `switch` para evitar errores. La sentencia `switch` ejecuta línea por línea (realmente, sentencia a sentencia). Al comienzo, no se ejecuta código. Sólo cuando se encuentra una sentencia `case` con un valor que coincide con el valor de la expresión `switch` PHP comienza a ejecutar las sentencias. PHP continúa ejecutando las sentencias hasta el final del bloque `switch`, o la primera vez que vea una sentencia `break`. Si no se escribe una sentencia `break` al final de una lista de sentencias case, PHP seguirá ejecutando las sentencias del siguiente case. Por ejemplo:

```
switch ($i) {
    case 0:
        print "i es igual a 0";
    case 1:
        print "i es igual a 1";
    case 2:
        print "i es igual a 2";
}
```

Aquí, si $i es igual a 0, ¡PHP ejecutaría todas las sentecias print! Si $i es igual a 1, PHP ejecutaría las últimas dos sentencias print y sólo si $i es igual a 2, se obtendría la conducta 'esperada' y solamente se mostraría 'i es igual a 2'. Así, es importante no olvidar las sentencias `break` (incluso aunque pueda querer evitar escribirlas intencionadamente en ciertas circunstancias).

En una sentencia `switch`, la condición se evalúa sólo una vez y el resultado se compara a cada sentencia `case`. En una sentencia `elseif`, la condición se evalúa otra vez. Si tu condición es más complicada que una comparación simple y/o está en un bucle estrecho, un `switch` puede ser más rápido.

La lista de sentencias de un case puede también estar vacía, lo cual simplemente pasa el control a la lista de sentencias del siguiente case.

```
switch ($i) {
    case 0:
    case 1:
    case 2:
        print "i es menor que 3, pero no negativo";
        break;
    case 3:
        print "i es 3";
}
```

Un case especial es el default case. Este case coincide con todo lo que no coincidan los otros case. Por ejemplo:

```
switch ($i) {
    case 0:
        print "i es igual a 0";
        break;
    case 1:
        print "i es igual a 1";
        break;
    case 2:
        print "i es igual a 2";
        break;
    default:
        print "i no es igual a 0, 1 o 2";
}
```

La expresión `case` puede ser cualquier expresión que se evalúe a un tipo simple, es decir, números enteros o de punto flotante y cadenas de texto. No se pueden usar aquí ni arrays ni objetos a menos que se conviertan a un tipo simple.

La sintaxis alternativa para las estructuras de control está también soportada con switch. Para más información, ver Sintaxis alternativa para estructuras de control.

```
switch ($i):
    case 0:
        print "i es igual 0";
        break;
    case 1:
        print "i es igual a 1";
        break;
    case 2:
        print "i es igual a 2";
        break;
    default:
        print "i no es igual a 0, 1 o 2";
endswitch;
```

require()

La sentencia require() se sustituye a sí misma con el archivo especificado, tal y como funciona la directiva `#include` de C.

Un punto importante sobre su funcionamiento es que cuando un archivo se incluye con include() o se requiere con require()), el intérprete sale del modo PHP y entra en modo HTML al principio del archivo referenciado, y vuelve de nuevo al modo PHP al final. Por esta razón, cualquier código dentro del archivo referenciado que debiera ser ejecutado como código PHP debe ser encerrado dentro de etiquetas válidas de comienzo y fin de PHP.

require() no es en realidad una función de PHP; es más una construcción del lenguaje. Está sujeta a algunas reglas distintas de las de funciones. Por ejemplo, require() no esta sujeto a ninguna estructura de control contenedora. Por otro lado, no devuelve ningún valor; intentar leer un valor de retorno de una llamada a un require() resulta en un error del intérprete.

A diferencia de include(), require() *siempre* leerá el archivo referenciado, *incluso si la línea en que está no se ejecuta nunca*. Si se quiere incluir condicionalmente un archivo, se usa include(). La sentencia conditional no afecta a require(). No obstante, si la línea en la cual aparece el require() no se ejecuta, tampoco se ejecutará el código del archivo referenciado.

De forma similar, las estructuras de bucle no afectan la conducta de require(). Aunque el código contenido en el archivo referenciado está todavía sujeto al bucle, el propio require() sólo ocurre una vez.

Esto significa que no se puede poner una sentencia require() dentro de una estructura de bucle y esperar que incluya el contenido de un archivo distinto en cada iteración. Para hacer esto, usa una sentencia include().

```
require( ’header.inc’ );
```

When a file is require()ed, the code it contains inherits the variable scope of the line on which the require() occurs. Any variables available at that line in the calling file will be available within the called file. If the require() occurs inside a function within the calling file, then all of the code contained in the called file will behave as though it had been defined inside that function.

If the require()ed file is called via HTTP using the fopen wrappers, and if the target server interprets the target file as PHP code, variables may be passed to the require()ed file using an URL request string as used with HTTP GET. This is not strictly speaking the same thing as require()ing the file and having it inherit the parent file’s variable scope; the script is actually being run on the remote server and the result is then being included into the local script.

```
/* This example assumes that someserver is configured to parse .php
 * files and not .txt files. Also, ’works’ here means that the variables
 * $varone and $vartwo are available within the require()ed file. */
```

```
/* Won't work; file.txt wasn't handled by someserver. */
require ("http://someserver/file.txt?varone=1&vartwo=2");

/* Won't work; looks for a file named 'file.php?varone=1&vartwo=2'
 * on the local filesystem. */
require ("file.php?varone=1&vartwo=2");

/* Works. */
require ("http://someserver/file.php?varone=1&vartwo=2");

$varone = 1;
$vartwo = 2;
require ("file.txt");  /* Works. */
require ("file.php");  /* Works. */
```

En PHP3, es posible ejecutar una sentencia `return` dentro de un archivo referenciado con require(), en tanto en cuanto esa sentencia aparezca en el ámbito global del archivo requerido (require()). No puede aparecer dentro de ningún bloque (lo que siginifica dentro de llaves({})). En PHP4, no obstante, esta capacidad ha sido desestimada. Si se necesita esta funcionalidad, véase include().

Ver tambien include(), require_once(), include_once(), readfile(), y virtual().

include()

La sentencia include() incluye y evalúa el archivo especificado.

Si "URL fopen wrappers" esta activada en PHP (como está en la configuración inicial), se puede especificar el fichero que se va a incluir usando una URL en vez de un fichero local (con su Path) Ver Ficheros remotos y fopen() para más información.

Un punto importante sobre su funcionamiento es que cuando un archivo se incluye con include() o se requiere con require(), el intérprete sale del modo PHP y entra en modo HTML al principio del archivo referenciado, y vuelve de nuevo al modo PHP al final. Por esta razón, cualquier código dentro del archivo referenciado que debiera ser ejecutado como código PHP debe ser encerrado dentro de etiquetas válidas de comienzo y fin de PHP.

Esto sucede cada vez que se encuentra la sentencia include(), así que se puede usar una sentencia include() dentro de una estructura de bucle para incluir un número de archivos diferentes.

```
$archivos = array ('primero.inc', 'segundo.inc', 'tercero.inc');
for ($i = 0; $i < count($archivos); $i++) {
    include $archivos[$i];
}
```

include() difiere de require() en que la sentencia include se re-evalúa cada vez que se encuentra (y sólo cuando está siendo ejecutada), mientras que la sentencia require() se reemplaza por el archivo

referenciado cuando se encuentra por primera vez, se vaya a evaluar el contenido del archivo o no (por ejemplo, si está dentro de una sentencia if cuya condición evaluada es falsa).

Debido a que include() es una construcción especial del lenguaje, se debe encerrar dentro de un bloque de sentencias si está dentro de un bloque condicional.

```
/* Esto es ERRÓNEO y no funcionará como se desea. */

if ($condicion)
    include($archivo);
else
    include($otro);

/* Esto es CORRECTO. */

if ($condicion) {
    include($archivo);
} else {
    include($otro);
}
```

En ambos, PHP3 y PHP4, es posible ejecutar una sentencia `return` dentro de un archivo incluido con include(), para terminar el procesado de ese archivo y volver al archivo de comandos que lo llamó. Existen algunas diferencias en el modo en que esto funciona, no obstante. La primera es que en PHP3, `return` no puede aparecer dentro de un bloque a menos que sea un bloque de función, en el cual `return` se aplica a esa función y no al archivo completo. En PHP4, no obstante, esta restricción no existe. También, PHP4 permite devolver valores desde archivos incluidos con include(). Se puede capturar el valor de la llamada a include() como se haría con una función normal. Esto genera un error de intérprete en PHP3.

Ejemplo 12-1. include() en PHP3 y PHP4

Asumamos la existencia del siguiente archivo (llamado `test.inc`) en el mismo directorio que el archivo principal:

```
<?php
echo "Antes del return <br>\n";
if ( 1 ) {
   return 27;
}
echo "Después del return <br>\n";
?>
```

Asumamos que el archivo principal (`main.html`) contiene lo siguiente:

```
<?php
$retval = include( ’test.inc’ );
echo "El archivo devolvió: ’$retval’<br>\n";
?>
```

Cuando se llama a `main.html` en PHP3, generará un error del intérprete en la linea 2; no se puede capturar el valor de un include() en PHP3. En PHP4, no obstante, el resultado será:

```
Antes del return
El archivo devolvió: ’27’
```

Ahora, asumamos que se ha modificado `main.html` para que contenga lo siguiente:

```
<?php
include( ’test.inc’ );
echo "De vuelta en main.html<br>\n";
?>
```

En PHP4, la salida será:

```
Antes del return
De vuelta en main.html
```

No obstante, PHP3 dará la siguiente salida:

```
Antes del return
27De vuelta en main.html

Parse error: parse error in /home/torben/public_html/phptest/main.html on line 5
```

El error del intérprete es resultado del hecho de que la sentencia `return` está encerrada en un bloque de no-función dentro de `test.inc`. Cuando el return se mueve fuera del bloque, la salida es:

```
Antes del return
27De vuelta en main.html
```

El ’27’ espúreo se debe al hecho de que PHP3 no soporta devolver valores con `return` desde archivos como ese.

When a file is include()ed, the code it contains inherits the variable scope of the line on which the include() occurs. Any variables available at that line in the calling file will be available within the called file. If the include() occurs inside a function within the calling file, then all of the code contained in the called file will behave as though it had been defined inside that function.

If the include()ed file is called via HTTP using the fopen wrappers, and if the target server interprets the target file as PHP code, variables may be passed to the include()ed file using an URL request string as used with HTTP GET. This is not strictly speaking the same thing as include()ing the file and having it inherit the parent file’s variable scope; the script is actually being run on the remote server and the result is then being included into the local script.

```
/* This example assumes that someserver is configured to parse .php
 * files and not .txt files. Also, ’works’ here means that the variables
 * $varone and $vartwo are available within the include()ed file. */

/* Won’t work; file.txt wasn’t handled by someserver. */
include ("http://someserver/file.txt?varone=1&vartwo=2");
```

```
/* Won't work; looks for a file named 'file.php?varone=1&vartwo=2'
 * on the local filesystem. */
include ("file.php?varone=1&vartwo=2");

/* Works. */
include ("http://someserver/file.php?varone=1&vartwo=2");

$varone = 1;
$vartwo = 2;
include ("file.txt");  /* Works. */
include ("file.php");  /* Works. */
```

See also require(), require_once(), include_once(), readfile(), and virtual().

require_once()

The require_once() statement replaces itself with the specified file, much like the C preprocessor's `#include` works, and in that respect is similar to the require() statement. The main difference is that in an inclusion chain, the use of require_once() will assure that the code is added to your script only once, and avoid clashes with variable values or function names that can happen.

For example, if you create the following 2 include files `utils.inc` and `foolib.inc`

Ejemplo 12-2. utils.inc

```
<?php
define(PHPVERSION, floor(phpversion()));
echo "GLOBALS ARE NICE\n";
function goodTea() {
        return "Oolong tea tastes good!";
}
?>
```

Ejemplo 12-3. foolib.inc

```
<?php
require ("utils.inc");
function showVar($var) {
        if (PHPVERSION == 4) {
                print_r($var);
        } else {
                dump_var($var);
        }
}

// bunch of other functions ...
?>
```

And then you write a script `cause_error_require.php`

Ejemplo 12-4. cause_error_require.php

```
<?php
require("foolib.inc");
/* the following will generate an error */
require("utils.inc");
$foo = array("1",array("complex","quaternion"));
echo "this is requiring utils.inc again which is also\n";
echo "required in foolib.inc\n";
echo "Running goodTea: ".goodTea()."\n";
echo "Printing foo: \n";
showVar($foo);
?>
```

When you try running the latter one, the resulting ouptut will be (using PHP 4.01pl2):

```
GLOBALS ARE NICE
GLOBALS ARE NICE

Fatal error:  Cannot redeclare causeerror() in utils.inc on line 5
```

By modifying `foolib.inc` and `cause_errror_require.php` to use require_once() instead of require() and renaming the last one to `avoid_error_require_once.php`, we have:

Ejemplo 12-5. foolib.inc (fixed)

```
...
require_once("utils.inc");
function showVar($var) {
...
```

Ejemplo 12-6. avoid_error_require_once.php

```
...
require_once("foolib.inc");
require_once("utils.inc");
$foo = array("1",array("complex","quaternion"));
...
```

And when running the latter, the output will be (using PHP 4.0.1pl2):

```
GLOBALS ARE NICE
```

```
this is requiring globals.inc again which is also
required in foolib.inc
Running goodTea: Oolong tea tastes good!
Printing foo:
Array
(
    [0] => 1
    [1] => Array
        (
            [0] => complex
            [1] => quaternion
        )

)
```

Also note that, analogous to the behavior of the `#include` of the C preprocessor, this statement acts at "compile time", e.g. when the script is parsed and before it is executed, and should not be used for parts of the script that need to be inserted dynamically during its execution. You should use include_once() or include() for that purpose.

For more examples on using require_once() and include_once(), look at the PEAR code included in the latest PHP source code distributions.

See also: require(), include(), include_once(), get_required_files(), get_included_files(), readfile(), and virtual().

include_once()

The include_once() statement includes and evaluates the specified file during the execution of the script. This is a behavior similar to the include() statement, with the important difference that if the code from a file has already been included, it will not be included again.

As mentioned in the require_once() description, the include_once() should be used in the cases in which the same file might be included and evaluated more than once during a particular execution of a script, and you want to be sure that it is included exactly once to avoid problems with function redefinitions, variable value reassignments, etc.

For more examples on using require_once() and include_once(), look at the PEAR code included in the latest PHP source code distributions.

See also: require(), include(), require_once(), get_required_files(), get_included_files(), readfile(), and virtual().

Capítulo 13. Funciones

Funciones definidas por el usuario

Una función se define con la siguiente sintaxis:

```
function foo ($arg_1, $arg_2, ..., $arg_n) {
    echo "Función de ejemplo.\n";
    return $retval;
}
```

Cualquier instrucción válida de PHP puede aparecer en el cuerpo de la función, incluso otras funiones y definiciones de clases.

En PHP3, las funciones deben definirse antes de que se referencien. En PHP4 no existe tal requerimiento.

PHP no soporta la sobrecarga de funciones, y tampoco es posible redefinir u ocultar funciones previamente declaradas.

PHP3 no soporta un número variable de parámetros, aunque sí soporta parámetros por defecto (ver Valores por defecto de de los parámetros para más información). PHP4 soporta ambos: ver Listas de longitud variable de parámetros y las referencias de las funciones func_num_args(), func_get_arg(), y func_get_args() para más información.

Parámetros de las funciones

La información puede suministrarse a las funciones mediante la lista de parámetros, una lista de variables y/o constantes separadas por comas.

PHP soporta pasar parámetros por valor (el comportamiento por defecto), por referencia, y parámetros por defecto. Listas de longitud variable de parámetros sólo están soportadas en PHP4 y posteriores; ver Listas de longitud variable de parámetros y la referencia de las funciones func_num_args(), func_get_arg(), y func_get_args() para más información. Un efecto similar puede conseguirse en PHP3 pasando un array de parámetros a la función:

```
function takes_array($input) {
    echo "$input[0] + $input[1] = ", $input[0]+$input[1];
}
```

Pasar parámetros por referencia

Por defecto, los parámetros de una función se pasan por valor (de manera que si cambias el valor del argumento dentro de la función, no se ve modificado fuera de ella). Si deseas permitir a una función modificar sus parámetros, debes pasarlos por referencia.

Si quieres que un parámetro de una función siempre se pase por referencia, puedes anteponer un ampersand (&) al nombre del parámetro en la definición de la función:

```
function add_some_extra(&$string) {
    $string .= ’ y algo más.’;
}
$str = ’Esto es una cadena, ’;
add_some_extra($str);
echo $str;    // Saca ’Esto es una cadena, y algo más.’
```

Si deseas pasar una variable por referencia a una función que no toma el parámetro por referencia por defecto, puedes anteponer un ampersand al nombre del parámetro en la llamada a la función:

```
function foo ($bar) {
    $bar .= ’ y algo más.’;
}
$str = ’Esto es una cadena, ’;
foo ($str);
echo $str;    // Saca ’Esto es una cadena, ’
foo (&$str);
echo $str;    // Saca ’Esto es una cadena, y algo más.’
```

Parámetros por defecto

Una función puede definir valores por defecto para los parámetros escalares estilo C++:

```
function makecoffee ($type = "cappucino") {
    return "Hacer una taza de $type.\n";
}
echo makecoffee ();
echo makecoffee ("espresso");
```

La salida del fragmento anterior es:

```
Hacer una taza de cappucino.
Hacer una taza de espresso.
```

El valor por defecto tiene que ser una expresión constante, y no una variable o miembro de una clase.

En PHP 4.0 también es posible especificar `unset` como parámetro por defecto. Esto significa que el argumento no tomará ningún valor en absoluto si el valor no es suministrado.

Destacar que cuando se usan parámetros por defecto, estos tienen que estar a la derecha de cualquier parámetro sin valor por defecto; de otra manera las cosas no funcionarán de la forma esperada. Considera el siguiente fragmento de código:

```
function makeyogurt ($type = "acidophilus", $flavour) {
    return "Haciendo un bol de $type $flavour.\n";
}

echo makeyogurt ("mora");   // No funcionará de la manera esperada
```

La salida del ejemplo anterior es:

```
Warning: Missing argument 2 in call to makeyogurt() in
/usr/local/etc/httpd/htdocs/php3test/functest.html on line 41
Haciendo un bol de mora.
```

Y ahora, compáralo con:

```
function makeyogurt ($flavour, $type = "acidophilus") {
    return "Haciendo un bol de $type $flavour.\n";
}

echo makeyogurt ("mora");   // funciona como se esperaba
```

La salida de este ejemplo es:

```
Haciendo un bol de acidophilus mora.
```

Lista de longitud variable de parámetros

PHP4 soporta las listas de longitud variable de parámetros en las funciones definidas por el usuario. Es realmente fácil, usando las funciones func_num_args(), func_get_arg(), y func_get_args().

No necesita de ninguna sintaxis especial, y las listas de parámetros pueden ser escritas en la llamada a la función y se comportarán de la manera esperada.

Devolver valores

Los valores se retornan usando la instrucción opcional return. Puede devolverse cualquier tipo de valor, incluyendo listas y objetos.

```
function square ($num) {
    return $num * $num;
}
echo square (4);   // saca '16'.
```

No puedes devolver múltiples valores desde una función, pero un efecto similar se puede conseguir devolviendo una lista.

```
function small_numbers() {
    return array (0, 1, 2);
}
list ($zero, $one, $two) = small_numbers();
```

`old_function`

La instrucción `old_function` permite declarar una función usando una sintaxis idéntica a la de PHP/FI2 (excepto que debes reemplazar 'function' por 'old_function').

Es una característica obsoleta, y debería ser usada únicamente por el conversor PHP/FI2->PHP3.

> **Aviso**
>
> Las funciones declaradas como `old_function` no pueden llamarse desde el código interno de PHP. Entre otras cosas, esto significa que no puedes usarlas en funciones como usort(), array_walk(), y register_shutdown_function(). Puedes solventar esta limitación escribiendo un "wrapper" (en PHP3 normal) que a su vez llame a la función declarada como `old_function`.

Funciones variable

PHP soporta el concepto de funciones variable, esto significa que si una variable tiene unos paréntesis añadidos al final, PHP buscará una función con el mismo nombre que la evaluación de la variable, e intentará ejecutarla. Entre otras cosas, esto te permite implementar retrollamadas (callbacks), tablas de funciones y demás.

Ejemplo 13-1. Ejemplo de función variable

```
<?php
function foo() {
    echo "Dentro de foo()<br>\n";
}

function bar( $arg = ” ) {
    echo "Dentro de bar(); el parámetro fue ’$arg’.<br>\n";
}

$func = ’foo’;
$func();
$func = ’bar’;
$func( ’test’ );
?>
```

Capítulo 14. Clases y Objetos

class

Una clase es una colección de variables y de funciones que acceden a esas variables. Una clase se define con la siguiente sintaxis:

```
<?php
class Cart {
    var $items;  // Items en nuestro carro de la compra

    // Añadir $num artículos de tipo $artnr al carro

    function add_item ($artnr, $num) {
        $this->items[$artnr] += $num;
    }

    // Sacar $num artículos del tipo $artnr del carro

    function remove_item ($artnr, $num) {
        if ($this->items[$artnr] > $num) {
            $this->items[$artnr] -= $num;
            return true;
        } else {
            return false;
        }
    }
}
?>
```

El ejemplo define una clase llamada Cart que consiste en un array asociativo de artículos en el carro y dos funciones para meter y sacar ítems del carro

Las clases son tipos, es decir, son plantillas para variables. Tienes que crear una variable del tipo deseado con el operador new.

```
 $cart = new Cart;
 $cart->add_item("10", 1);
```

Este ejemplo crea un objeto $cart de clase Cart. La función add_item() de ese objeto se llama para añadir un item del artículo número 10 al carro.

Las Clases pueden ser extensiones de otras clases. Las clases extendidas o derivadas tienen todas las variables y funciones de la clase base y lo que les añadas al extender la definición. La herencia múltiple no está soportada.

```
class Named_Cart extends Cart {
    var $owner;

    function set_owner ($name) {
        $this->owner = $name;
```

```
    }
}
```

Ese ejemplo define una clase Named_Cart (carro con nombre o dueño) que tiene todas las variables y funciones de Cart, y además añade la variable $owner y una función adicional set_owner(). Un carro con nombre se crea de la forma habitual y, una vez hecho, puedes acceder al propietario del carro. En los carros con nombre también puedes acceder a las funciones normales del carro:

```
$ncart = new Named_Cart;    // Creamos un carro con nombre
$ncart->set_owner ("kris"); // Nombramos el carro
print $ncart->owner;        // Imprimimos el nombre del propietario
$ncart->add_item ("10", 1); // Funcionalidad heredada de Cart
```

Entre funciones de una clase, la variable $this hace referencia al propio objeto. Tienes que usar $this->loquesea para acceder a una variable o función llamada loquesea del objeto actual.

Los constructores son funciones de una clase que se llaman automáticamente al crear una nueva instancia (objeto) de una clase. Una función se convierte en constructor cuando tiene el mismo nombre que la clase.

```
class Auto_Cart extends Cart {
    function Auto_Cart () {
        $this->add_item ("10", 1);
    }
}
```

Este ejemplo define una clase Auto_Cart que es un Cart junto con un constructor que inicializa el carro con un item del tipo de artículo "10" cada vez que se crea un nuevo Auto_Cart con "new". Los constructores también pueden recibir parámetros y estos parámetros pueden ser opcionales, lo que los hace más útiles.

```
class Constructor_Cart extends Cart {
    function Constructor_Cart ($item = "10", $num = 1) {
        $this->add_item ($item, $num);
    }
}

// Compramos las mismas cosas aburridas de siempre

$default_cart   = new Constructor_Cart;

// Compramos las cosas interesantes

$different_cart = new Constructor_Cart ("20", 17);
```

Atención

Para las clases derivadas, el constructor de la clase padre no es llamado automáticamente cuando se llama al constructor de la clase derivada.

Capítulo 15. References Explained